# Sistema de Inteligencia Artificial para Videojuego Action RPG 2D

*Carlos Alejandro Goncevatt*

Director: Antonio Chica Calaf
Especialidad: Computación


Universitat Politècnica de Catalunya

Fecha: 16/10/2018

# Índice

# 1. Introducción

## 1.1 Resumen

En este proyecto se han implementado el prototipo de un videojuego Action RPG 2D de cámara isométrica, y 3 modelos de inteligencia artificial para el mismo. El objetivo era introducirse en el desarrollo de videojuegos, así como poder evaluar que tan aplicable es, a nivel básico, el aprendizaje automático en este tipo de proyectos.

Para ello primero se ha comenzado por pensar el concepto del juego y diseñarlo. El motor ha sido implementado casi desde 0, de modo que los sprites animados, las entidades del juego, la gestión de escena y el sistema de colisiones, entre otras funcionalidades, han sido implementados y adaptados a los conceptos y el diseño planteados, dentro del mínimo necesitado para tener un prototipo jugable.

En el apartado del sistema de IA, primero ha sido necesario investigar y aprender sobre los métodos de aprendizaje automático, para así poder decidir el más adecuado para lo que buscamos hacer. Ya decidida la metodología se implementaron 3 modelos, 2 de ellos basados en scripting. El primero enfocándose en un comportamiento directo y simple. El segundo, también de scripting, intenta crear un comportamiento más complejo, valorando muchos otros elementos del entorno y utilizando más movimientos, incluso contando con algunas ventajas. El tercero tiene la capacidad de aprender de las acciones del jugador con el fin de imitarlo, comienza con un conjunto de acciones y estados iniciales, los cuales luego irá completando con nuevos estados o nuevas acciones que aprenderá cada vez que el jugador luche contra otro personaje.

Finalmente se realizan pruebas de usuario para poder valorar el rendimiento de las implementaciones y si la IA con aprendizaje ha sido exitosa en su objetivo de imitar al jugador.

## 1.2 Organización de la Memoria

En este documento se intentan definir y explicar las distintas partes que conforman el proyecto, así como el proceso de realización de las mismas. Esta primera sección de Introducción incluye un resumen del trabajo realizado y los resultados, así como un resumen de la organización y el contenido de la memoria del proyecto.

En la segunda sección se encuentran el contexto sobre el que se desarrolla el proyecto. Se formula el problema que busca solucionar el trabajo, así como se establecen los actores implicados y el alcance del proyecto. A su vez se hace un repaso de los campos que cubre el trabajo y su estado actual.

En la tercera sección se desarrolla en más profundidad sobre el diseño del juego y la implementación realizada. En el primer caso se intentan plantear de forma clara las ideas y conceptos en que se basa el juego, así como se nombran y comentan varios títulos de referencia que influenciaron el proceso de creación. En la segunda parte se describen

apartados más técnicos sobre cómo se han implementado algunas de las partes más relevantes del motor.

La cuarta sección se centra en el diseño e implementación de la Inteligencia Artificial, comentando las ideas y conceptos que se han intentado modelar, así como las implementaciones finales. También se hace un breve análisis sobre posibles cambios o variantes que se podrían haber realizado.

La quinta sección muestra y analiza los datos extraídos de las pruebas y encuestas de usuarios.

La sexta sección desarrolla la conclusión del trabajo.

La séptima sección intenta plantear algunos puntos sobre los cuales se puede trabajar para continuar el trabajo iniciado en este proyecto, ya sea por el lado del desarrollo del juego, así como del sistema de inteligencia artificial.

La octava, y última, sección describe todo lo relacionado a la gestión del proyecto, su organización, planificación y el análisis de sostenibilidad.

# 2. Alcance del Proyecto y Contextualización

## 2.1 Contextualización

Para este proyecto es importante identificar y analizar el contexto en el que se desarrolla. Hay dos áreas principales que conforman los pilares de este trabajo y lo engloban en su totalidad: los videojuegos y la Inteligencia Artificial en los mismos.

La historia de los videojuegos es extensa y se remonta hasta la década de 1950. En esta época se puede resaltar el experimento *"Tennis for Two"* de William Higinbotham, en el cual se demostraba el control interactivo de un juego en pantalla. Aunque hoy en día muchos no lo consideran como un videojuego real [1], se lo puede ver como un precursor de los mismos a nivel conceptual.

Continuando en el tiempo se pueden resaltar otros tantos puntos claves durante el origen de los videojuegos: como la creación de *Spacewar!* En 1962 o el lanzamiento de *Pong* en 1972, el cual fue un gran éxito y marcó las bases para una industria del videojuego exitosa [1]. Con el pasar de los años la industria se fue extendiendo, surgieron las consolas de juegos para los hogares y los ordenadores comenzaron a ser accesibles al público, las personas podía disfrutar del entretenimiento e incluso comenzar a desarrollar sus propios videojuegos. Con el pasar de los años fueron surgiendo personajes icónicos, como Mario, de *Super Mario Bros.* (1985) [1], o Sonic, de *Sonic the Hedgehog* (1991) [1]; así como gran variedad de juegos: como *Pac-Man* (1979) [1], *Donkey Kong* (1981) [2], *Q*bert* (1982) [1] y *Tetris* (1984) [3], entre tantos otros.

Todos estos juegos se pueden clasificar en gran variedad de géneros, los elementos de cada uno de ellos se han logrado mezclar para dar forma a nuevos tipos. Éstos están definidos por diversas características, como los controles, la cámara, la interacción del jugador con las demás entidades del mundo o la manera de contar la historia. Algunos de estos géneros son: RPG (Role Playing Game o Juegos de Rol), Novelas Visuales, Acción, Aventura, Horror, Rogue-like o Action RPG (una combinación de elementos RPG y Acción). También se puede hablar de una clasificación más particular dependiendo de si el juego utiliza gráficos en 2D (2 dimensiones), 3D o 2.5D (juego en 2D que simula 3D), incluso se pueden considerar juegos en 1D, Realidad Aumentada o Realidad Virtual, estos dos últimos soportados con gráficos en 3D.

La complejidad de los juegos ha incrementado enormemente en comparación a los orígenes, hoy en día cada lanzamiento puede incluir una gran cantidad de mecánicas variables, una historia extensa, personajes, música, gráficos, efectos visuales, y muchos elementos más. La cantidad de personas implicadas en la creación puede variar según las dimensiones del proyecto, desde tener 1 solo desarrollador, que lo hace todo, a varios grupos que se encargan de distintos aspectos durante el proceso y cuyos trabajos se unen para dar forma al producto final. Esto se refleja en los costes y tiempo dedicado al desarrollo, así como las ganancias y el dinero que mueve la industria, por ejemplo el récord de 12.5$ mil millones de dólares que hizo la industria del videojuego de Estados Unidos en 2006 [1].

Podemos ver que los videojuegos, como tales, han pasado por muchas cosas y hoy en día son uno de los principales medios de entretenimiento, la industria del videojuego continúa creciendo, existe mucho interés por parte del público en participar de este mercado,

ya sea como desarrolladores, editores o publicistas, e incluso han conseguido reconocimiento en el campo deportivo con los eSports [4].

En el campo de la Inteligencia Artificial dentro de los videojuegos hay detalles particulares. Los primeros juegos creados estaban enfocados en ofrecer entretenimiento basándose en enfrentar a dos jugadores humanos, cada uno controlaba una entidad y jugaban el uno contra el otro en igualdad de condiciones, juegos como *Spacewar*! o *Pong*, no contaban con IA. Probablemente esto se daba por las limitaciones de los sistemas de la época, así como los problemas de accesibilidad a las tecnologías de desarrollo y lo nuevo que era este campo de trabajo. Con el paso del tiempo se comenzaron a crear videojuegos de un solo jugador, en éstos la persona debía progresar en el juego realizando algunas acciones específicas o enfrentándose a personajes no jugadores con comportamientos predefinidos, si bien esto no es exactamente Inteligencia Artificial, es lo que se denominó como tal dentro de este campo.

La IA en los videojuegos se esconde bajo la superficie del mismo, el sistema que controla a los personajes no jugadores, y parte del entorno, ayuda a "dar vida" al mundo en el que se sumerge al jugador, determina la manera en que él interactúa con el juego [5]. La IA en los videojuegos cubre una variada cantidad de técnicas y algoritmos, tanto de este campo de estudio, como de otros. En la Game Developers Conference de 2013, Alex J. Champandard dijo: "Mi predicción es un poco controversial. Creo que el próximo gran salto de la IA en los juegos es, en realidad, inteligencia artificial" [5]. Esta frase la dice en el contexto de que buena parte de los desarrolladores de IA en videojuegos ven lo que hacen como algo externo al campo técnico de la inteligencia artificial, algunos hasta dicen que tiene muy poca relación con las técnicas avanzadas de IA que salen de las organizaciones académicas e industriales [5].

El desafío de los sistemas de inteligencia artificial en los videojuegos está vinculada a la experiencia del jugador, es decir que está diseñada para transmitir una experiencia que provea diversión, drama, algún tipo de divertimento o emociones que dependen del momento [5]. Por ello se considera que la IA está diseñada para ser tan inteligente como se necesite para dotar de una buena experiencia al jugador, así como, para muchos juegos, se debe comportar de manera predecible y, en casos, apenas inteligente [5]. La idea es mantener el comportamiento dentro de un campo controlable.

Los objetivos de los desarrolladores de IA avanzados, fuera del entorno de los videojuegos, van más por el camino de crear sistemas que puedan imitar la verdadera inteligencia biológica [5]. Particularmente, el caso de Machine Learning (ML) incluye desafíos como crear una IA con la capacidad de aprender y adaptarse a nuevos entornos; crear sistemas robustos para entender el habla y el ruido; extraer patrones o desarrollar una IA que pueda evolucionar y hacer frente a tareas no triviales [5].

El uso de técnicas como el ML en los videojuegos no es muy normal, esto se debe principalmente a que su implementación no es sencilla y muchas veces el comportamiento de estos sistemas puede ser impredecible, lo cual podría poner en riesgo la experiencia del jugador: no es cuestión de que no se pueda obtener un buen comportamiento de las técnicas de aprendizaje automático, sino que puede no ser el comportamiento deseado para la experiencia del jugador que se busca [5].

### Actores Implicados

En este proyecto están implicados varios actores:

Desarrollador: Carlos Alejandro Goncevatt. Es la persona que se encargará de analizar, investigar, documentar e implementar todo el proyecto. Encargada de gestionar el proyecto y comunicarse con el director del mismo, así como de cumplir con las fechas límites de las entregas.

Director del Proyecto: Antonio Chica Calaf. Supervisa el proyecto para que siga el ritmo correcto y cumpla con los objetivos. También puede ayudar y aportar consejo al desarrollador para realizar el proyecto.

Diseñador y Artista: Marcelo Nahuel Goncevatt. Es la persona encargada de la mayoría del diseño del juego, sus mecánicas, personajes e historias. También será el artista que dibuje los escenarios, personajes, efectos, interfaz y demás elementos necesarios para el juego.

Beta-testers: son las personas que realizarán pruebas del juego y su IA para poder verificar su funcionamiento. Son necesarios para poder detectar errores y corregirlos.

Público: son las personas a las que el juego pueda ir dirigido, ya sea por la búsqueda del desafío, así como porque simplemente les guste el género del mismo. En general, cualquier persona a la que le puedan gustar estos tipos de videojuegos.

## 2.2 Formulación del Problema

Hoy en día existen muchos ejemplos de juegos que han aplicado técnicas más complejas de IA en sus mecánicas o funcionalidades. Los jugadores de distintos tipos de juegos buscan entornos más creíbles, así como desafíos y muchas veces estos aspectos son influenciados por el comportamiento y las interacciones de los personajes no jugadores. Los métodos para intentar contentar al público son variados, la manera más común de encarar la cuestión de manera segura es con el uso de la programación del comportamiento predefinido y controlado, dotando a las entidades de la IA con ventajas sobre el jugador.

La idea del proyecto es poder crear un videojuego de género Action RPG 2D que enfrente al jugador a distintos enemigos, éstos tendrán un sistema de IA que combine el comportamiento estático con el dinámico, a través del uso de scripting y conceptos de Machine Learning para influenciar el comportamiento, de modo que los personajes no jugadores tengan la capacidad de aprender y adaptarse al jugador, cambiando sus comportamientos y acciones en consecuencia. De esta forma se espera poder crear un desafío para la persona sin que las entidades cuenten con ventajas de números, como cantidad excesiva de vida o daño, o más información de la que necesite el sistema para funcionar. A su vez se espera poder ofrecer una batalla final contra una entidad cuyo comportamiento esté completamente basado en el del propio jugador, es decir que sus movimientos y estilo se asemejen a los que la propia persona haya usado a lo largo del juego.

### Objetivos del Proyecto

Los objetivos de este proyecto son aquellas cuestiones que deben lograr ser cubiertas para poder considerar que el sistema implementado soluciona el problema descrito en la sección 3.2.

La primera cuestión a encarar es la de desarrollar el videojuego, o prototipo, para el cual desarrollaremos el sistema de IA. Es necesario tener un sistema sobre el cual trabajar para conocer la información y parámetros disponibles.

El segundo punto será el de investigar y estudiar los distintos métodos de aprendizaje automático, permitiéndonos obtener una visión global y analizar las ventajas y desventajas de cada uno, de modo que podamos utilizar el algoritmo más adecuado.

El tercer objetivo es el desarrollo del propio sistema de IA para el juego, utilizando y adaptando las funcionalidades necesarias del método elegido anteriormente.

Finalmente, necesitaremos comprobar, a través de distintos medios, si las entidades se comportan de la manera deseada y si el juego provee al jugador de la experiencia buscada.

## 2.3 Estado del Arte

La Inteligencia Artificial es un programa o sistema que se encarga de controlar la toma de decisiones y el comportamiento de alguna entidad. Dentro del ámbito de los videojuegos se utiliza para simular comportamiento inteligente por parte de los personajes no jugadores, o de los enemigos a los que se enfrenta el jugador. Uno de las métodos más conocidos y utilizados en este ámbito es la creación de comportamientos ad-hoc, dentro del cual se pueden definir tres técnicas diferentes [6]:

Las máquinas de estados finitos, las cuales están definidas por tres componentes principales: un número de estados que tienen información sobre alguna tarea; un número de transiciones entre estados que indican un cambio de estado y están descritas por una condición que debe cumplirse; y un conjunto de acciones que deben realizarse dentro de cada estado. La máquina solo puede estar en un estado a la vez, a su vez el estado actual puede cambiar a otro si la condición de la transición correspondiente se cumple [6].

Los árboles de decisiones, los cuales modelan transiciones entre un conjunto finito de tareas (o decisiones). Emplean una estructura de árbol con un nodo raíz y un número de nodos padres, con sus correspondientes hijos, que representan decisiones. Se recorre el árbol comenzando desde la raíz, se procede a activar la ejecución de los pares padre-hijos tal y como se denotan en la estructura; un nodo hijo puede retornar uno de tres valores en un momento de tiempo predeterminado: *run, success* o *failure* [6].

La IA basada en utilidad, en la que las distintas instancias de un juego tienen asignadas una función de utilidad que les da un valor. Este sistema considera todas las utilidades disponibles para un agente y las opciones que tiene, y de entre ellas decide cual es la más importante a considerar en el momento de la evaluación [6].

Estas técnicas son relativamente simples, aunque pueden crecer enormemente en complejidad dependiendo de lo que se quiera hacer, y se mantienen dentro de un entorno más controlado. Dependiendo de las condiciones del entorno pueden mostrar comportamientos variados, pero todo debe haber sido predefinido en el sistema de IA de la entidad, por lo que se pueden considerar estas técnicas como algo "estático".

Dentro del campo "dinámico" encontraremos las técnicas de Machine Learning, aprendizaje automático. De los métodos de aprendizaje supervisado nos enfocaremos en las redes neuronales artificiales (ANN – Artificial Neural Networks). El aprendizaje supervisado requiere de un conjunto de ejemplos etiquetados para entrenarse, su objetivo no es solo aprender de las parejas de entradas-salidas, sino también derivar una función que aproximara la relación entre ellos. Esta función debería ser capaz de aplicarse correctamente a instancias nunca antes vistas de parejas de entradas-salidas, una propiedad llamada generalización [6].

Una red neuronal artificial es una aproximación bio-inspirada para la inteligencia computacional y el aprendizaje automático. Es un conjunto de unidades de proceso (neuronas) interconectadas, las cuales fueron originalmente diseñadas para modelar la manera en que un cerebro biológico procesa información, opera, aprende y realiza varias tareas. Varios tipos de redes neuronales son aplicables para análisis de la regresión, clasificación, aprendizaje con preferencia, e incluso aprendizaje no supervisado. [6]

Otro de los métodos de aprendizaje automático que capta el interés de este proyecto es el del aprendizaje por refuerzo, una técnica inspirada en la psicología del comportamiento y cómo, en particular, los humanos y animales aprenden a tomar decisiones por medio de recompensas (positivas o negativas) recibidas por parte del entorno. En este tipo de aprendizaje usualmente no se dispone de ejemplos de "buen" comportamiento; sino que la señal de entrenamiento del algoritmo es proveída por el entorno basándose en cómo el agente interactúa con él. Dicho de otra forma: en un punto particular en el tiempo $t$, el agente se encuentra en un estado $s$ y decide llevar a cabo la acción $a$ de todas las disponibles en su estado actual. Como respuesta el entorno envía una recompensa inmediata $r$. Es a través de las interacciones continuas con su entorno, que el agente gradualmente aprende a elegir acciones que maximizan la suma de sus recompensas. [6]

Dentro del aprendizaje por refuerzo nos enfocaremos en el algoritmo del Q-Learning, el cual es un algoritmo de aprendizaje de diferencia temporal que se basa en una representación tabular de valores. Informalmente cada valor $Q(s, a)$ representa que tan bueno es elegir la acción $a$ en el estado $s$. El agente de Q-Learning aprende de la experiencia seleccionando acciones y recibiendo recompensas a través de la estimación de que tan bueno es un estado, basado en que tan bueno pensamos que es el estado siguiente (llamado bootstrapping, se actualiza una estimación basándose en otra estimación). El objetivo del agente es maximizar su recompensa estimada escogiendo las acciones correctas en cada estado. [6]

Esta técnica tiene unas cuantas limitaciones asociadas, mayoritariamente, con la forma de su representación. Aún así existe un tipo de híbrido de técnicas que puede utilizarse para abordar esta cuestión: utilizar una red neuronal artificial como aproximador del valor, de manera tal que se reemplaza la tabla. De esta manera es posible aplicar el algoritmo a espacios de representaciones acción-estado de mayor tamaño. [6]

Han habido varios trabajos de investigación alrededor de la Inteligencia Artificial Adaptativa para Videojuegos, en estos se han realizado propuestas y pruebas utilizando distintas técnicas e implementaciones sobre varias categorías de juegos. Una serie de experimentos comprobó que una determinada aproximación para establecer una IA de juego "rápidamente adaptable" para el videojuego *Spring* fue capaz de obtener victoria de manera eficaz, así como mantener estados de empate durante tiempos relativamente largos [7]. Otro trabajo de investigación fue capaz de demostrar la viabilidad del aprendizaje online en juegos de pelea, así como el dominio de éstos juegos propone problemas únicos en los que algunas técnicas estándar de aprendizaje automático resultan poco prácticas [8]. Un tercer trabajo introdujo tres propuestas de investigación orientadas a la creación de sistemas de IA capaces de adaptarse al jugador y resultar más agradables para el mismo, a través de la aplicación de técnicas de razonamiento basadas en casos [9]. Otra investigación permitió comprobar la viabilidad del Dynamic Scripting para el desarrollo de inteligencias artificiales adaptativas al cumplir con los requerimientos de velocidad, eficacia, robustez, claridad, variedad, eficiencia, consistencia y escalabilidad [10]. Un último trabajo a comentar es aquel en el que se prueba que una IA con capacidad de adaptación mejora la experiencia del jugador, así como la personalidad de la persona también es un factor a considerar para predecir el tipo de IA adaptativa con la cual disfrutará más [11].

Podemos concluir que existen muchas técnicas que pueden ser utilizadas para implementar un sistema de inteligencia artificial para un juego, nos interesa considerar las clásicas y las de aprendizaje automático. Dentro de estas últimas han habido trabajos que han buscado analizar la viabilidad, a nivel de funcionamiento y resultados, de varios de estos métodos en distintos tipos de juego. Todas estas investigaciones han ofrecido resultados que nos permiten valorar de manera positiva su aplicación, además de dar pistas de en que apartados es posible expandir los estudios. En nuestro caso trabajaremos con un género de juego que aún no se ha revisado en profundidad, el Action RPG, para el cual implementaremos diferentes tipos de IA, aplicando y modificando técnicas de los tipos comentados anteriormente.

## 2.4 Alcance

Este proyecto consta de dos partes bien diferenciables, pero relacionadas: el desarrollo del videojuego y la implementación del sistema de IA.

En primer lugar se encarará la implementación del videojuego. El desarrollador se reunirá con el diseñador para poder determinar el tipo de juego que debe hacerse, su estilo, cámara, gráficos y mecánicas. La idea es que ambas personas sepan que elementos nos encontraremos y cómo interactuarán éstos, ya sea entre ellos o con el jugador. Una vez definido y diseñado el juego cada uno podrá comenzar con sus respectivas tareas.

El primer paso en la implementación son las bases del juego, el objetivo es conseguir un prototipo. Se deben crear los elementos y funcionalidades básicas, las primeras entidades, el movimiento, las animaciones, el escenario y cualquier tipo de funcionalidad auxiliar que pueda necesitarse. Al acabar esta primera parte se puede continuar con otros aspectos del juego, trabajando más en la interfaz y los textos. Con todo esto implementado a un nivel básico se tendría una base para el prototipo. Con el tiempo se continuarán terminando de

pulir algunas de estas funcionalidades según sea necesario.

Teniendo un prototipo básico y funcional comenzaremos a pensar en el sistema de IA, para ello lo primero será analizar y estudiar los algoritmos de aprendizaje automático que puedan sernos de utilidad según nuestras necesidades, lo más seguro es que será algo vinculado al aprendizaje por refuerzo (Reinforcement Learning). Habiendo visto las posibilidades, sus ventajas, desventajas y características elegiremos el método que más nos beneficie.

Ya habiendo pensado en la metodología para la parte dinámica de la inteligencia artificial pasaremos a implementar la parte estática, ya que es la más sencilla y nos permitirá comenzar a hacer pruebas controladas de las interacciones del jugador con los personajes no jugadores hostiles. Se deberán implementar maneras para que las entidades se muevan, ataquen, se protejan, esquiven y realicen acciones en intervalos de tiempo específicos. Una vez finalizada la parte estática continuaremos por implementar y adaptar todo lo relacionado a la capacidad de aprendizaje y adaptación. Se realizarán tantas pruebas e iteraciones como sea necesario, principalmente en la segunda parte del sistema, ya que es el más complicado.

Mientras se progrese de manera general con el desarrollo se irán añadiendo los elementos visuales que vayan siendo terminados por el Artista. Una vez terminada la IA se continuará haciendo pruebas con la misma, así como se refinará el juego para intentar acercarlo más a un estado de producto terminado, dentro de lo posible y con los elementos disponibles. Este proceso final incluirá el añadido de eventos que ayuden a progresar con la historia y el guión.

# 3. El Juego

## 3.1 Diseño

### Ideas y Concepto

Existen dos objetivos bases a partir de los cuales se centran los planteos, ideas y conceptos para construir este videojuego. El primer objetivo es el de presentar una jugabilidad rápida y dinámica, intentando hacer énfasis en una acción continua para mantener al jugador atento a su entorno cuando se dé la situación. El segundo es el de la experiencia, buscamos poder crear un mundo en el cual se desarrolle una historia en la cual será participe el jugador, intentando conectar todos los elementos del entorno de manera que no se pierda la inmersión. Considerando ambos objetivos se definió el género más adecuado para este videojuego, el cual es el Action Role-Playing Game, conocido como Action RPG o ARPG. Este sub-género de los juegos de rol (Role-Playing Game o RPG) representa estas ideas perfectamente, ya que hace énfasis en sistemas de combate en tiempo real, similares a los de los juegos de acción del tipo "Hack and Slash"[1]. Al mismo tiempo se pueden incorporar otros tipos de mecánicas y sistemas para mejorar la inmersión en el mundo (interactuando con las entidades que lo habitan) o la percepción de crecimiento del jugador, como estadísticas del personaje, incremento de niveles o misiones.

Definir el estilo visual del juego requiere valorar varias cuestiones, entre ellas la viabilidad para poder crear el material necesario. Considerando los conocimientos, preferencias personales y capacidades de los miembros del equipo que realizan el proyecto, se decidió hacer el videojuego en 2D con animaciones fotograma por fotograma (frame per frame), de modo que sea más sencillo crear el motor desde cero, así como los recursos externos (texturas e imágenes). A su vez se ha tomado la decisión de utilizar una cámara isométrica, para añadir un efecto de profundidad y aumentar las posibilidades de movimiento teniendo ocho direcciones posibles para moverse (las cuatro alineadas a los ejes y las diagonales). Esto aumenta la complejidad a nivel de desarrollo pero permite integrar de mejor manera los objetivos anteriormente mencionados, ya que se puede crear un mundo continuo que pueda mezclar e intercalar los espacios de progreso de historia e interacción, con los de combates sin tener que sufrir cambios de cámara drásticos o pantallas de carga de transición.

### Combate

Como fue mencionado en el apartado 3.1.1, se busca plantear un sistema de combate rápido y dinámico, que recompense al jugador por su habilidad y por estar atento al entorno, sabiendo aprovechar todos los elementos disponibles. Para ello se ha pensado en un modelo de combate en tiempo real con variedad de movimientos, tanto defensivos como ofensivos.

Comenzando por los movimientos ofensivos habrán dos "acciones" de ataque, las básicas serán separadas, cuando sea posible, por los lados que se utilizarán para realizar dicha acción, es decir, que el jugador podrá utilizar ataques con, por ejemplo, los brazos

1 Un tipo de jugabilidad enfocada en el combate, en juegos de acción suele implicar un enfoque orientado al combate cuerpo a cuerpo en tiempo real.

izquierdo y derecho de forma separada. Esto está pensado para dar más opciones y proporcionar un mayor control en la ofensiva, lo que puede permitir explotar algún punto vulnerable del enemigo. Como se comentó antes, no todos los ataques estarán separados por lados, algunos son “comunes”, en otras palabras, no importa si se use la acción ofensiva izquierda o derecha, el movimiento resultante será el mismo. Entre la variedad de ataques habrá de varios rangos, ya sea corto, medio o largo, proporcionando una amplia variedad de posibilidades que puedan ajustarse al estilo del jugador, como un estilo agresivo cuerpo a cuerpo o un estilo más precavido que mantiene una distancia prudencial y reduce al enemigo con ataques a distancia.

Por otro lado encontraremos movimientos defensivos. En esta variedad también habrán aquellos diferenciados por el lado sobre el cual se realizan (a modo de contra-acción sobre los ataques). Tendremos acciones defensivas básicas como bloquear o repeler, si se realiza en el momento adecuado, una determinada acción ofensiva, dejando vulnerable al atacante. A su vez dispondremos de acciones de movimiento rápido que permitan realizar evasiones o recuperarse de un estado negativo, otra posibilidad es usar estos movimientos para aumentar o reducir la distancia que separa a los combatientes de manera casi instantánea.

Tal y como se comentó antes, el jugador también podrá moverse en ocho direcciones en el plano para poder posicionarse según lo que requiera, como por ejemplo para intentar tomar distancia, al ritmo que permita la velocidad de movimiento del personaje, o buscar un lado en el que el enemigo sea vulnerable.

## Condiciones de Victoria y Derrota

En un videojuego es necesario especificar cuáles son las condiciones de victoria y derrota. En el primer caso porque debe haber un objetivo a cumplir, el jugador debe intentar conseguir algo y ser recompensado positivamente por ello. En el segundo caso porque un juego en que no se puede perder no tiene sentido, no hay un sentimiento de riesgo si no se puede perder, el jugador necesita saber que debe esforzarse por evitar que una determinada situación o condición negativa se dé para evitar recibir un “castigo”.

En este juego la condición de derrota será la “muerte” del personaje, es decir cuando el jugador pierda todos sus puntos de salud, ya sea por consecuencia de un combate o algún evento particular, como una trampa en el terreno. Esta condición se mantiene en todo momento del juego.

Por otro lado, podemos determinar dos condiciones de victoria claras: una particular para la situación de combate o enfrentamiento y otra, a mayor escala y a largo plazo, para el propio juego. La primera solo puede darse cuando el jugador está enfrentándose a una entidad enemiga, en este caso la condición de victoria es lograr reducir los puntos de salud del enemigo a cero, de modo que logra derrotarlo. La segunda condición se aplica al juego y su historia, por ello es considerada a mayor escala y a largo plazo. Se puede considerar que el jugador ha ganado el juego cuando derrota al jefe final y acaba con el último evento definido en la historia.

## Títulos de Referencia

Hoy en día existe una gran cantidad y variedad de videojuegos, muchos pueden conseguir fama y renombre por innovar con nuevas mecánicas, o plantear algún sistema e implementarlo con una perspectiva y forma que no se había visto anteriormente. Por esto se pueden ver muchas versiones, interpretaciones e implementaciones de un mismo tipo de combate o jugabilidad, cada una con sus características, tanto negativas como positivas. A continuación se listan y comentan de algunos títulos de referencia que han influenciado en el proceso de diseño de este videojuego.

### Battlerite

Es un juego de acción por equipos con una jugabilidad basada en el género MOBA (Multiplayer Online Battle Arena) y desarrollado por Stunlock Studios [15]. Este juego ha tenido efecto en la definición del sistema de combate y en el tipo de cámara. En el apartado de batallas es un juego con cierto dinamismo, disponen de gran variedad de personajes, todos poseen una habilidad de movilidad, que puede usarse de forma defensiva y ofensiva, y se pueden dividir por roles, teniendo distintas habilidades y estadísticas que suelen relacionarse al mismo. Por ejemplo, el rol de cuerpo a cuerpo suele tener más salud y ataques de corta distancia; por otro lado el rol de rango tiene ataques y habilidades de larga distancia, aunque suelen tener menos resistencia. Cada rol suele implicar su propio estilo de juego, no tendría sentido usar un personaje de rango para atacar a distancia cuerpo a cuerpo.


*Fig. 1: Batalla 2vs2 en Battlerite.*

### Melty Blood

Es una saga de juegos de lucha desarrollado por Type Moon y FRENCH-BREAD, basado en el universo de la novela visual Tsukihime de Type Moon. Posee muchas de las características de los juegos de combate de estilo arcade, entre los que se puede destacar la capacidad de realizar gran cantidad de combinaciones de ataques a partir de los ataques base de los que disponen los personajes, y en conjunto con los movimientos, como saltos e impulsos. Este tipo de batallas 1vs1 con variedad de posibilidades, ataques y estilos es una de las cosas que buscamos conseguir en nuestro sistema de combate, ampliado a las dimensiones de nuestro juego. Otro elemento que influencia nuestro videojuego es el estilo

visual, ya que es un juego 2D, aunque con una cámara distinta, con animación fotograma por fotograma que nos permite tener un tipo de referencia de como crear el efecto de movimiento en las imágenes que conforman la animación de una determinada acción.


*Fig. 2: Melty Blood Actress Again.*

## Naruto Shippuden: Narutimate Accel 2

Es la quinta entrega de la saga de juegos de lucha desarrollado por Cyberconnect2 y publicada por Namco Bandai, basado en el manga y anime Naruto, de Masashi Kishimoto. Es un videojuego en 3D que utiliza una perspectiva similar a los juegos de lucha 2D. Su sistema de combate es bastante dinámico y veloz, utiliza un modelo "smash button", es decir que los ataques suelen realizarse en base a presionar un botón de "acción" acompañado de algún otro como modificador, por ejemplo un botón de movimiento en alguna dirección. A su vez, los personajes disponen de movimientos para cargar sobre su enemigo y, utilizado en el momento adecuado, pueden evitar un ataque moviéndose instantáneamente a la espalda del oponente. Este es otro de los juegos de combate que, realizándolo exclusivamente 1vs1, influenció nuestros conceptos sobre el "combate ideal" que buscamos conseguir en nuestro juego.

*Fig. 3: Instante antes de un choque de técnicas en Narutimate Accel 2.*

## Hyper Light Drifter

Es un Action RPG 2D desarrollado por Heart Machine y publicado en 2016. Es uno de los juegos que más influenció las ideas y conceptos de este proyecto, tanto de combate, género, estilo gráfico y cámara. Las batallas en este juego son rápidas e intensas, los enemigos suelen abundar y los terrenos varían, hay que hacer buen uso de las distintas técnicas, tanto ofensivas como defensivas, para poder sobrevivir y derrotar a todos los oponentes. La atención del jugador y el correcto uso de la evasión es algo vital para poder superar los desafíos, características que también buscamos en nuestro sistema.

Considerando tanto combates, como modo historia y entornos, es una buena representación de lo que queremos conseguir, fundir todos los elementos y situaciones según sea necesario, de modo que todo sea continuo y el jugador pueda experimentar la historia, así como enfrentar los obstáculos sin que haya una transición intermedia que pueda romper la inmersión. Este juego ha sido de los más influyentes al momento de valorar el tipo de cámara para utilizar, ya que era una gran demostración del uso de la cámara isométrica en un videojuego en 2D.

*Fig. 4: Batalla contra el primer jefe de Hyper Light Drifter.*

## World of Warcraft

Es un MMORPG (Massively Multiplayer Online Role-Playing Game) desarrollado por Blizzard Entertainment y publicado en 2004. Es un juego con muchos elementos y un mundo muy extenso. Influenció en nuestra manera de plantear el progreso del jugador en el modo historia, así como nos ha permitido ver como las distintas cámaras pueden tener efectos variados en la experiencia e inmersión del jugador, ya sea por el ángulo respecto al personaje, así como la distancia del mismo. Aún así se han debido considerar las restricciones que existían al estar viendo estos efectos en un juego en 3D, siendo que nuestro juego es en 2D.


*Fig. 5: Cámara de un jugador en una arena de 5vs5 en World of Warcraft.*

Devil May Cry

Es una serie de videojuegos de acción, shooter y Hack and Slash desarrollados y publicados por Capcom. Desde la salida del primer juego de la serie en 2001 se ha convertido en un referente de su género, por tener un combate pulido, rápido y dinámico, que proporciona gran variedad de armas y técnicas que el jugador puede combinar para conseguir derrotar a sus enemigos de manera variada y satisfactoria. Es otro de los grandes referentes que consideramos al momento de pensar y diseñar nuestro sistema de combate, valorando la variedad de posibles movimientos y de cómo combinarlos para obtener enfrentamiento llamativos, dinámicos e impactantes. Otra de las características que se pueden resaltar es el sistema de estilo, que es una manera de puntuar al jugador por variar sus ataques, derrotar a los enemigos de manera eficiente y sin recibir daño, mientras más tiempo se mantenga un alto nivel de estilo, más “dinero” recibe el jugador.


*Fig. 6: Combate final 1vs1 entre Dante y Vergil en Devil May Cry 3.*


*Fig. 7: Jugador repeliendo ataque enemigo en Devil May Cry 4.*

NieR: Automata

Es un videojuego Action RPG desarrollado por Yoko Taro y PlatinumGames, y publicado por Square Enix [16]. Es otra influencia sobre el sistema de combate, ya que tiene una acción intensa y veloz, el jugador debe estar atento a los ataques enemigos que pueden venir de diversas direcciones simultáneamente. Es un juego que recompensa al jugador por responder en el momento idóneo de forma impactante y visualmente gratificante.

*Fig. 8: 2B (jugador) contra varios enemigos al mismo tiempo.*

Dead Cells

Es otro de los juegos que han influenciado el estilo visual y el combate, dispone de gran variedad de armas y técnicas, el combate hace gran énfasis en la atención al entorno y enemigos, así como la capacidad del jugador de reaccionar ante lo que ocurra. Este juego también nos permitió valorar otras posibilidades de puntos de vista para nuestro juego, valorando los puntos positivos y negativos del género de acción y plataformas, tal y como Hyper Light Drifter (3.1.4.4) nos lo permitió con la cámara isométrica.

*Fig. 9: Jugador atacando a un enemigo arquero.*

## Skullgirls

Es un videojuego de lucha en 2D desarrollado por Lab Zero Games [17]. Ha tenido principal influencia en el diseño y estilo visual, ya que demuestra el nivel de calidad que se puede alcanzar utilizando animación 2D fotograma por fotograma. Cada personaje y escenario está dibujado y animado de manera detallada y minuciosa, manteniendo consistente el estilo visual y el arte del juego.


*Fig. 10: Captura de animación de ataque y bloqueo durante un combate.*

## Shadow of the Colossus

Es un videojuego de acción y aventura desarrollado por Sony Computer Entertainment. Es muy reconocido por su planteo innovador a nivel de mecánicas y jugabilidad. El jugador deberá enfrentar a diversos jefes colosos en batallas de 1vs1, en las cuales deberá escalar por el cuerpo de su enemigo y atacar a los puntos débiles. Una de las

acciones únicas que incluye es la reserva de un comando para la acción de sujetarse a su objetivo, de modo que el jugador debía explícitamente mantener esa orden para poder escalar, así como podía soltarse si lo necesitaba (o se le acababa la energía). Este planteo es lo que inspiró la decisión de separar las acciones para proporcionar más recursos al jugador, aumentando el rango de opciones posibles a realizar.

*Fig. 11: Wander (jugador) sobre la cabeza de un coloso.*

## 3.2 Implementación

En los siguientes apartados se describirán las implementaciones de algunas de las partes más relevantes del motor del juego y del prototipo.

### Gestión de Recursos Externos

Los recursos externos son archivos de diversos formatos y tipos que son utilizados por el juego, nuestro sistema actualmente es capaz de preparar imágenes, sonidos y fuentes. Para gestionar y tener los recursos accesibles en la aplicación definimos una clase template llamada *ResourceHolder,* Este contenedor es capaz de cargar un recurso definido en una ruta y asociarle un identificador especificado por parámetro, en caso de no encontrar el archivo indica el error para que pueda ser verificado. También puede buscar en su estructura un recurso con un identificador especificado por parámetro y, en caso de encontrarlo, retornar un puntero al mismo. Finalmente se pueden borrar recursos guardados en el contenedor para poder liberar memoria utilizada por la aplicación. Por ser una clase template es necesario definir el tipo de recurso e identificador que se debe usar para guardar la información.

```
typedef ResourceHolder<sf::Texture, std::string> TextureHolder;
typedef ResourceHolder<sf::Font, std::string> FontHolder;
typedef ResourceHolder<sf::Sound, std::string> SoundHolder;
```

*Fig. 12: Definición de tipos de "holders" para texturas, fuentes y sonidos.*

Para cargar los recursos se creó un clase estática *ResourceLoader* que tiene definidas distintas funciones para cargar recursos, estas funciones pueden recibir un identificador (en caso de ser para cargar recursos individuales), la ruta al archivo y el contenedor correspondiente. Esta clase está pensada para tener funciones particulares que contengan el conjunto de instrucciones para cargar grupos de recursos según sea necesario, un ejemplo de esto se puede ver en la función *LoadTest* la cual tiene especificadas las llamadas necesarias con los parámetros para cargar en memoria los recursos utilizados en el prototipo.

## Sprites Animados

Una de las características más relevantes a desarrollar son los sprites animados, es decir las estructuras que permitan a una entidad cambiar de fotograma con las especificaciones deseadas. Para ello definimos una clase *AnimationData*, la cual guardará toda la información referente a una determinada animación: cantidad de frames, cajas de colisiones (se explicarán posteriormente), información de movimiento, texturas, coordenadas de origen de las correspondientes texturas, tiempo de actualización y si la animación puede detectar colisiones de manera constante.

*AnimationData* inicializa un vector en el cual guardaré, en orden correspondiente, punteros a las texturas almacenadas en el *TextureHolder*. De la misma manera tendrá otros vectores en los cuales se guarda la demás información (mencionada anteriormente) correspondiente a cada frame. De esta forma se puede guardar la información de cada animación de forma independiente y acceder a la misma según sea necesario.

Las animaciones que existen en el juego, y prototipo, están definidas en una estructura *enum* llamada *Animation*. Todas las animaciones describen un "estado" y para cada uno hay 8 animaciones, que se corresponden a las 8 direcciones a las que puede encarar o moverse la entidad en este mundo: abajo (D), diagonal izquierda hacia abajo (DD), horizontal izquierda (H), diagonal izquierda hacia arriba (DU), arriba (U), diagonal derecha hacia arriba (DUR), horizontal derecha (HR) y diagonal derecha hacia abajo (DDR). Las animaciones se describen como *ANIM_[STATE]_[DIRECTION]*.

```
enum Animation
{
    // General group
    // Idle
    ANIM_IDLE_D = 0, // Down
    ANIM_IDLE_DD, // Diagonal Down
    ANIM_IDLE_H, // Horizontal
    ANIM_IDLE_DU, // Diagonal Up
    ANIM_IDLE_U, // Up

    ANIM_IDLE_DUR, // Right anim for non mirrored sprites
    ANIM_IDLE_HR,
    ANIM_IDLE_DDR, // End of Idle state

    // Run
    ANIM_RUN_START_D,
    ANIM_RUN_START_DD,
    ANIM_RUN_START_H,
    ANIM_RUN_START_DU,
    ANIM_RUN_START_U,
    ANIM_RUN_START_DUR,
    ANIM_RUN_START_HR,
    ANIM_RUN_START_DDR,
```

*Fig. 13: Definiciones de Idle y Run Start.*

Ya teniendo una clase para mantener la información de cada animación y sus correspondientes frames, procedemos a definir la clase *SpriteData* la cual representará un sprite animado. Esta clase hereda de la clase *sf::Sprite* de SFML para que pueda ser dibujada de forma simple en la ventana. *SpriteData* guarda las animaciones de las que dispone en un mapa cuyo identificador es del tipo *Animation*, por ejemplo la animación por defecto para toda entidad es la que se corresponde al 0: *ANIM_IDLE_D*. Los sprites animados tienen definidas muchas funcionalidades, pueden ejecutar animaciones consecutivas utilizando una estructura de *queue*, pueden ejecutar las animaciones en reversa, pueden ejecutar animaciones con una velocidad determinada (distinta de la original) y pueden ejecutar una determinada cantidad de iteraciones antes de desactivar su renderizado. Estos sprites también pueden ejecutar "modificadores" definidos por la clase *SpriteMod*, la cual aplica un tipo de transformación dependiendo del tipo (*AlphaMod* modifica el factor alfa, *XTranslation* / *YTranslation* aplican cambios en las coordenadas y *XScale* / *YScale* modifican la escala).

```
private:
    std::unordered_map<Animation, AnimationData> m_AnimationMap;
    std::queue<std::pair<Animation, std::pair<sf::Uint32, bool>>> m_AnimationQueue;
    std::list<SpriteMod*> m_SpriteMods;

    TextureHolder& m_Holder;
    AnimationData* m_ActualAnimationData;

    Animation m_ActualAnimation, m_DefaultAnimation;
```

*Fig. 14: Algunas variables de SpriteData*

# Entidades

En las siguientes sub-secciones se describen y comentan las implementaciones de los distintos tipos de entidades que se pueden encontrar de forma activa en el prototipo. También se describen los métodos para actualizar las entidades de la escena, así como el renderizado de las mismas.

## Objetos

La clase *Object* es la que define un objeto genérico del mundo. Estos objetos tienen una gran cantidad de características y funcionalidades. En primer lugar todos tienen un sprite animado en el cual se deben registrar las animaciones que le correspondan al objeto.

```
m_LoadingScreen = new Object("Loading_BG", *this, m_Textures, "", m_Scene.GetLayer(LAYER_LOADING_SCREEN));
AnimationData lbgData(1);
lbgData.SetFrameTexture(&m_Textures.Get("Loading_BG"), 0, m_LoadingScreen->GetSprite().GetDefaultUpdateTime());
m_LoadingScreen->GetSprite().InsertAnimation(ANIM_IDLE_D, lbgData);
m_LoadingScreen->GetSprite().EnableAnimation();
```

*Fig. 15: Creación del objeto estático de pantalla de carga.*

En esta clase se pueden especificar comandos o cadenas de comandos, es decir que a cada objeto se le pueden asignar acciones a realizar en consecuencia a uno o más inputs, esto permite dotar a todas las entidades de la capacidad de procesar y actuar según estos datos, con lo que, por ejemplo, el jugador podría tomar control de cualquier entidad y ésta respondería a los inputs de teclado que tenga especificados. Esta clase también puede procesar el movimiento en el plano y mantiene la información sobre la dirección en la que

puede estar desplazándose la entidad. Otra de las funcionalidades que tienen los objetos es la de realizar eventos luego de que transcurra una determinada cantidad de tiempo, lo que permite crear distintos comportamientos o series de eventos según lo que busque el desarrollador.

```
bool Object::HandleCommandInput(Command cmd, sf::Uint32 duration)
{
    m_Input.push_back(std::make_pair(duration, cmd));

    // Check Command Sequences
    for (auto seqs = m_SequenceCommands.begin(); seqs != m_SequenceCommands.end(); ++seqs)
        if ((*seqs).first.CheckSequence(m_Input))
        {
            m_Input.clear();
            return (*seqs).second.Execute(this);
        }

    // Check commands based on Animation
    auto command = m_Commands.find(cmd);

    if (command != m_Commands.end())
    {
        auto c = command->second.find(m_Sprite.GetCurrentAnimationId());

        if (c != command->second.end() && c->second.CheckFrame(m_Sprite.GetActualFrame()))
            return c->second.Execute(this);
    }

    return false;
}
```

*Fig. 16: Función que procesa comandos y ejecuta acciones si es necesario.*

La clase *Object* también guarda información sobre la capa en la que es dibujada la entidad, así como si solo está activa o también debe ser dibujada. A través del objeto se pueden realizar cambios u obtener información sobre el sprite, también se pueden obtener distintos datos de relevancia, como la distancia euclidiana (al cuadrado) hasta un punto o entidad, el estado en el que se encuentra, la dirección hacia la que está mirando o moviéndose, entre otras cosas. Esta clase también guarda un puntero al objeto que controla la inteligencia artificial (en caso de que lo tenga) e implementa las funcionalidades para que la propia instancia sea destruida, o para crear nuevas entidades y añadirlas al mundo.

Esta es la clase base para todas las entidades y sirve para representar elementos del juego que no son parte del mundo o entorno con el que interactúa el jugador una vez toma control del personaje principal, por ejemplo elementos de la interfaz o decoraciones del mundo.


*Fig. 17: Objeto de la pantalla de carga (personaje con cartel de "Loading"). Los 3 puntos son otro objeto animado con 3 frames (en cada frame se añade un punto).*

## Objetos del Mundo

La clase *WorldObject* es la primera que hereda de *Object* y añade atributos y funcionalidades orientadas a tratar con entidades que pueden interactuar entre sí dentro del mundo. En esta clase se registra información como la salud de la entidad, la cual define si está viva o muerta, y el objetivo, otra entidad probablemente enemiga, sobre el que se centrará. También procesa las consecuencias de las interacciones con otras entidades, como recibir daño.

```
    virtual void ProcessDmg();
    virtual void RecievedDamage();

protected:
    float m_DmgMod;
    WorldObject* m_Target;
    sf::Int32 m_MaxHealth, m_CurHealth;
    sf::Uint32 m_Dmg;
```

*Fig. 18: Algunos miembros de la clase WorldObject: función para procesar si la entidad recibió daño; función que ejecuta las consecuencias de recibir daño; variables de la clase.*

## Dummy

Esta clase hereda de *WorldObject* para crear una entidad simple con la cual puede interactuar el jugador una vez toma control de su personaje. Las instancias de la clase *Dummy* solo crearán un muñeco con 2 animaciones y unas pocas cajas de colisiones, a su vez éstas no pueden morir, su función es la de ser un muñeco de entrenamiento.

```
Dummy::Dummy(std::string id, Game& game, TextureHolder& holder, std::string path, Layer* layer, sf::Int32 health) :
    WorldObject(id, game, holder, path, layer, health)
{
    m_Sprite.SetDefaultUpdateTime(TimePerFrame.asMilliseconds());

    sf::RectangleShape s(sf::Vector2f(215.f, 195.f));
    s.setOutlineThickness(10.f);
    s.setFillColor(sf::Color::Transparent);
    s.setOrigin(s.getSize().x / 2.f, s.getSize().y / 2.f);

    AnimationData standd(1);
    standd.SetFrameTexture(&holder.Get("Dummy_Stand_D"), 0, m_Sprite.GetDefaultUpdateTime());
    standd.SetTerrainCollider(TerrainBox(s, sf::Vector2f(1245.f, 1505.f)), 0);

    AnimationData damaged(8);
    for (int i = 0; i < 8; ++i)
    {
        damaged.SetFrameTexture(&holder.Get("Dummy_Damaged_D_" + std::to_string(i)), i, m_Sprite.GetDefaultUpdateTime());
        damaged.SetTerrainCollider(TerrainBox(s, sf::Vector2f(1245.f, 1505.f)), i);
    }

    // Add Hurtbox
    standd.SetHurtboxesAmount(0, 1);
    std::vector<CollisionBox>& hb = standd.GetHurtboxes(0);
    hb[0].SetAction(std::function<void(Object*, Object*)>([] (Object* owner, Object* actor) {
        owner->GetSpriteData().PlayAnimation(ANIM_DAMAGED_D);
    }));
    hb[0].SetLocalPosition(sf::Vector2f(1229.f, 1298.f));

    sf::RectangleShape r(sf::Vector2f(200.f, 265.f));
    r.setFillColor(sf::Color::Transparent);
    r.setOutlineThickness(10.f);
    r.setOrigin(r.getSize().x / 2.f, r.getSize().y / 2.f);
    hb[0].SetBoundingBox(r);

    m_Sprite.InsertAnimation(ANIM_IDLE_D, standd);
    m_Sprite.InsertAnimation(ANIM_DAMAGED_D, damaged);

    m_Sprite.EnableAnimation();
}
```

std::function<void (Object *, Object *)>::fur

+ 10 overloads

*Fig. 19: Constructor de la clase Dummy.*

En la figura 19 se puede observar el constructor de esta clase, en la cual se define el tiempo de actualización del objeto, se crea una caja a modo de plantilla, se construyen las animaciones “IDLE_D” y “DAMAGED_D”, se les añaden las cajas de colisiones, se añaden las animaciones al sprite animado y, finalmente, se activa el sprite para que muestre la textura correspondiente, como puede verse a continuación, en la figura 20.

*Fig. 20: Dummy in-game.*

## Jugador y Enemigo

La clase *Player* se encarga de construir el personaje jugable de este prototipo, el cual será utilizado por el jugador y también será el enemigo, es decir, que ambos se enfrentarán con el mismo conjunto de movimientos. El constructor se encarga de crear las animaciones, asignando todo lo necesario: texturas, cajas de colisiones e interacciones. También registra todos los movimientos que pueden ser utilizados, así como gestiona las limitaciones de los mismos (como por ejemplo el tiempo de enfriamiento de los saltos).

```
Player::Player(std::string id, Game& game, TextureHolder& holder, std::string path, Layer* layer) : WorldObject(id, game, holder, path, layer, 100)
{
    if (id.compare("Test_Enemy") == 0)
    {
        sf::Color c = m_Sprite.getColor();
        c.g = 155;
        c.b = 155;

        m_Sprite.setColor(c);
    }

    m_JabCombo = true;

    // PH ---------------------------------------------------------------------------------------------------------------
    m_Sprite.SetMirroredMovement(false);
    m_Sprite.SetDefaultUpdateTime(static_cast<int>((1.f / 18.f) * IN_MILLISECONDS));

    LoadIdleAnimations(holder);
    LoadRunAnimations(holder);
    LoadJumpMoveAnimations(holder);
    LoadBasicAttackAnimations(holder);
    LoadDefenseAnimations(holder);
    LoadDashesAnimations(holder);
    LoadStunnedAnimations(holder);

    m_Sprite.EnableAnimation();

    LoadBasicAttackCommands();
    LoadMoveCommands();
    LoadJumpCommands();
    LoadDefenseCommands();
    LoadDashesCommands();

    m_JumpInCD = m_JumpOutCD = m_DashCD = 0;
}
```

*Fig. 21: Constructor de la clase Player.*

En la figura 21 se puede ver el constructor de la clase, en el cual son observables las llamadas a las funciones que crean y registran las animaciones y los comandos. También se puede ver una modificación al color del sprite del objeto en caso de que sea el "enemigo" (de modo que sea fácil distinguirlo del jugador), otras inicializaciones son los tiempos de enfriamiento (cooldown o CD) y el booleano que verifica si el jugador puede lanzar una combinación rápida de puñetazos (primero con un brazo y luego con el otro).

Esta entidad dispone de varios movimientos:

- Ataque: un puñetazo rápido lanzado en la dirección a la que mira el jugador, hay 2 tipos: izquierdo y derecho. Cuando se lanza uno de los 2 ataques, se puede encadenar un golpe con el otro si se realiza la acción en el marco de tiempo correcto; aunque esto no puede encadenarse de manera indefinida.
- Salto: un comando que mueve instantáneamente al personaje. Hay 2 versiones, la primera es el "jump in", el cual cierra distancias colocando a la entidad a rango cuerpo a cuerpo, si se está fuera de este rango; la segunda es el "jump out", el cual mueve al jugador instantáneamente al perímetro exterior de la arena que se encuentra a espaldas del jugador (en relación a su oponente). Cada uno tiene un tiempo de enfriamiento individual de 30 segundos.
- Movimiento: comandos básicos para comenzar la animación de movimiento en

alguna dirección, son 4 tipos que se corresponden a las direcciones alineadas a los ejes, las direcciones diagonales se consiguen combinando los comandos pertinentes.

- Bloqueo: el personaje toma una postura defensiva y bloquea completamente los ataques que vengan desde la dirección a la que se encuentra mirando.
- Rechazo: estando en estado de bloqueo se pueden utilizar los comandos de ataque para intentar rechazar algún ataque. Hay 2 tipos: derecho e izquierdo, los cuales rechazan ataques con el brazo izquierdo y derecho respectivamente.
- Carrera (Dash): un movimiento de alta velocidad que puede usarse para reducir o incrementar distancias entre los personajes, se pueden utilizar hasta 3 consecutivos y tienen un tiempo de enfriamiento de 5 segundos (conteo iniciado cada vez que se realiza la acción, las 3 cargas se recuperan al pasar ese tiempo). Solo se puede utilizar mientras el jugador se mueve en una dirección y puede encadenarse con el el final de un dash previo para continuar en la misma, o en otra, trayectoria.

## Update

La actualización de las entidades activas en el juego se realiza de forma constante en el bucle principal del juego. Todas ellas se encuentran registradas en una capa de la escena la cual mantiene las instancias "vivas", a su vez cada capa se actualiza en un orden predeterminado por la escena tal y como se puede ver en la figura 22.

```
while (m_Window.isOpen())
{
    timeSinceLastUpdate += clock.restart();

    while (timeSinceLastUpdate > TimePerFrame)
    {
        CheckLoadingState();
        ProcessEvents();

        Update(TimePerFrame);

        timeSinceLastUpdate -= TimePerFrame;
    }

    Render();
}
```

```
void Scene::Update(sf::Uint32 diff)
{
    for (Layer& l : m_Layers)
        l.UpdateObjects(diff);
}
```

*Fig. 22: Game loop y función de Update de Scene (llamada en la función Update de Game)*

## Render

Una vez el tiempo entre frames pasa se procede a realizar el renderizado del estado actual del juego. En este caso se limpia la ventana actual dejándola en negro y se procede a pintar la escena. Tal y como en el proceso de actualización, el renderizado de cada capa se realiza según el orden que tienen predefinido, la única diferencia se da con la capa principal en la que se encuentran las entidades que interactúan en el mundo, ya que en este punto el orden en que deben renderizarse los objetos está dictado por su posición en el eje Y en el terreno (definida por la Terrainbox, que será explicada en la sección 3.2.5.3). De esta forma se puede simular la sensación de profundidad, que un elemento está por delante de otro.

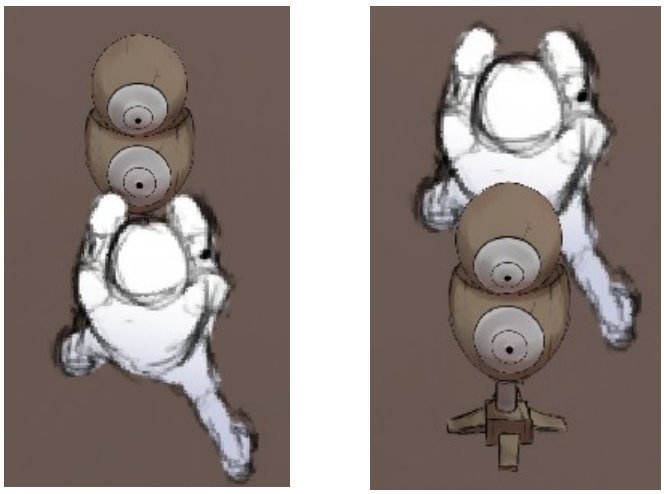


*Fig. 23: (Izq.) Jugador en frente del dummy. (Der.) Jugador detrás del dummy.*

# Comandos y Controles

## Mapeo de Teclas

Para poder crear un sistema que permita modificar las configuraciones de teclas se decidió crear la definición de *Command*, la cual representa un tipo de comando o instrucción que puede ser recibido y procesado por una entidad. Usando este concepto el juego, de forma inicial, relaciona determinadas teclas (u otras acciones, como los botones del mouse) a comandos específicos, estas asignaciones pueden ser modificadas por el usuario otorgando la funcionalidad de controles configurables (aunque el prototipo no tiene implementada una interfaz para hacer esto último).

```
enum Command
{
    CMD_UP          = 1,
    CMD_DOWN,
    CMD_LEFT,
    CMD_RIGHT,
    CMD_ATTACK_1,
    CMD_ATTACK_2,

    CMD_JUMP_IN,
    CMD_JUMP_OUT,

    CMD_ACTION,

    CMD_BLOCK,
    CMD_DASH,

    CMD_MAX
};
```

```
void Game::LoadDefaultCommands()
{
    m_KeyMap.insert(std::make_pair<sf::Keyboard::Key, Command>(sf::Keyboard::W, CMD_UP));
    m_KeyMap.insert(std::make_pair<sf::Keyboard::Key, Command>(sf::Keyboard::S, CMD_DOWN));
    m_KeyMap.insert(std::make_pair<sf::Keyboard::Key, Command>(sf::Keyboard::A, CMD_LEFT));
    m_KeyMap.insert(std::make_pair<sf::Keyboard::Key, Command>(sf::Keyboard::D, CMD_RIGHT));
    m_KeyMap.insert(std::make_pair<sf::Keyboard::Key, Command>(sf::Keyboard::Q, CMD_JUMP_IN));
    m_KeyMap.insert(std::make_pair<sf::Keyboard::Key, Command>(sf::Keyboard::E, CMD_JUMP_OUT));
    m_MouseMap.insert(std::make_pair<sf::Mouse::Button, Command>(sf::Mouse::Button::Left, CMD_ATTACK_1));
    m_MouseMap.insert(std::make_pair<sf::Mouse::Button, Command>(sf::Mouse::Button::Right, CMD_ATTACK_2));
    m_KeyMap.insert(std::make_pair<sf::Keyboard::Key, Command>(sf::Keyboard::LShift, CMD_BLOCK));
    m_KeyMap.insert(std::make_pair<sf::Keyboard::Key, Command>(sf::Keyboard::Return, CMD_ACTION));
    m_KeyMap.insert(std::make_pair<sf::Keyboard::Key, Command>(sf::Keyboard::Space, CMD_DASH));

    // SaveCommandsConfig();
}
```

*Fig. 24: (Izq.) Definición de comandos habilitados en el prototipo.(Der.) Función que vincula las teclas a sus respectivos comandos.*

### Gestión de Inputs

Para poder procesar los inputs realizados por el jugador se decidió realizar un simple método de polling, de modo que antes de proceder a la actualización de las entidades en cada iteración del bucle del juego se llama a la función *ProcessEvents*, la cual se encarga de verificar todos los eventos capturados por la ventana y procesarlos de forma iterativa.

```
void Game::ProcessEvents()
{
    sf::Event event;

    while (m_Window.pollEvent(event))
    {
        // Call event handler depending on event type
        /***
        E.g.

        if (event.type == sf::Event::KeyPressed)
            handleInput(event);
        else if (event.type == sf::Event::MouseMoved)
            handleHover(event);
        ***/
        if (event.type == sf::Event::KeyPressed)
            HandleKeyPressed(event);
        else if (event.type == sf::Event::MouseButtonPressed)
            HandleMouseButtonPressed(event);
        else if (event.type == sf::Event::KeyReleased)
            HandleKeyReleased(event);

        if (event.type == sf::Event::Closed) // PH
            m_Window.close();
    }
}
```

*Fig. 25: Función ProcessEvents que es llamada en cada iteración del game loop.*

## Sistema de Colisiones

### Cajas de Colisiones

Considerando el tipo de combates que buscamos crear en el juego se decidió preparar un sistema de colisiones que, de base, permitiera tener tantas regiones con capacidad de colisión, como fuesen necesarias, así como que estas regiones pudiesen ser ajustadas tanto como fuese necesario a los personajes. Se definió que las regiones serían rectángulos, cajas, a las que se le pudiesen definir el centro, las dimensiones y la rotación. Además de estos datos básicos, las cajas de colisiones disponen de otros parámetros como “tipo” y “valor” (aunque han quedado en desuso en el prototipo), y las funciones que deben ser ejecutadas por las entidades involucradas en la verificación. Todas estas características están definidas en la clase base *CollisionBox*, con la cual se definen las hitboxes y hurtboxes, así como de la cual hereda la clase *TerrainBox*. Todas estas cajas serán comentadas en los siguientes apartados.

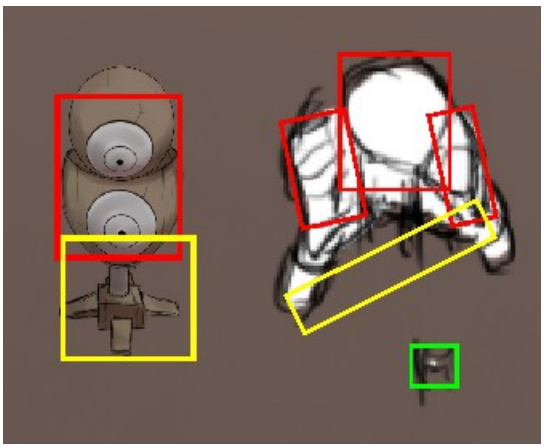


*Fig. 26: Cajas de colisiones del dummy y del jugador. (Rojo) Hurtboxes. (Verde) Hitboxes. (Amarillo) Terrainboxes.*

## Hitboxes y Hurtboxes

Las hitboxes son cajas de colisiones que permiten a una entidad interactuar con otra, por ejemplo, son las cajas que permiten que un personaje golpee a otro, causando daño. Por otro lado, las hurtboxes son las cajas "objetivo" de las hitboxes, son aquellas con la que una entidad detecta y recibe una interacción de otra. En la figura 26 puede verse un ejemplo de la hitbox y las hurtboxes de un frame de la animación de ataque del personaje jugador.

## Terrainbox

La clase *TerrainBox* hereda de *CollisionBox* y se utiliza de 2 maneras: el primer caso es como una caja definida como parte del terreno, marcando un área particular (como, por ejemplo, un obstáculo o una trampa); el segundo caso es una caja única definida para un frame de una animación, donde representa la posición del personaje en el mundo (ya que el ángulo de la cámara no permite utilizar la propia posición del sprite como una referencia fiable). Las *TerrainBox* solo pueden colisionar con otras *TerrainBox*, por lo que son usadas para todo lo que tenga que ver con la interacción de un personaje con el terreno. Es posible ver como los centros de las terrainboxes de la figura 26 intentan marcar el centro de la posición del personaje proyectada en el suelo, así como sus dimensiones intentan marcar el área que puede considerarse "ocupada" por la entidad.

## Verificación de Colisiones

Las verificaciones de colisiones se realizan de forma separada para las cajas que interactúan y solo se llevan a cabo para las entidades de la capa principal. Durante la actualización se verifican las *TerrainBox* de cada entidad con las del terreno. Esto se debe a que las animaciones son las que gestionan la velocidad de movimiento, por lo que una vez el sprite de una entidad es actualizado se aprovecha para verificar si intenta realizar un movimiento válido o si entra en contacto con un área especial del terreno.

Por otro lado, las verificaciones de colisiones entre entidades se realizan una vez todas han sido actualizadas. Cada entidad se evalúa con todas las demás, en cada iteración se

verifica si hay una colisión solo en una dirección (las hitboxes de A colisionan con las hurtboxes de B, o viceversa) o si hay una doble colisión (las hitboxes de A colisionan con las hurtboxes de B y las hitboxes de B colisionan con las hurtboxes de A). Una vez se tiene el resultado se procede a ejecutar las acciones pertinentes según el caso que se dé. Cada par de entidades solo se evalúa una vez por iteración del game loop.

```
firstSec = obj1->CollidesWith(obj2, hitFirst, hurtSec);
secFirst = obj2->CollidesWith(obj1, hitSec, hurtFirst);

if (firstSec && secFirst)
{
    obj1->SetCollided(true);
    obj2->SetCollided(true);

    hitFirst.ExecuteClashAction(obj1, obj2);
    hitSec.ExecuteClashAction(obj2, obj1);
}
else if (firstSec)
{
    obj1->SetCollided(true);

    hitFirst.ExecuteAction(obj1, obj2);
    hurtSec.ExecuteAction(obj2, obj1);
}
else if (secFirst)
{
    obj2->SetCollided(true);

    hitSec.ExecuteAction(obj2, obj1);
    hurtFirst.ExecuteAction(obj1, obj2);
}
```

*Fig. 27: Verificación de colisiones y ejecución de acciones pertinentes de la escena.*

## Escenas

Dentro del prototipo se pueden distinguir 2 escenas con las que el jugador podrá interactuar.

### Menú Principal

La pantalla de menú principal (figura 28) se compone de una imagen principal y 2 botones con los que puede interactuar el jugador, el botón a seleccionar será el que tenga el borde resaltado en rojo. Para desplazarse es posible usar los comandos “Up” (tecla 'W') y “Down” (tecla 'S'), así como el comando “Action” (tecla 'Enter') para seleccionar la opción marcada. La opción “*Start*” iniciará el prototipo, en caso de no estar cargados los recursos mostrará una pantalla de carga (fig. 17), si están preparados entonces pasará a la escena de la zona de combate. Por otro lado, “*Exit*” cerrará la aplicación y finalizará el proceso.


*Fig. 28: Pantalla de menú principal.*

## Zona de Combate

En la figura 29 tenemos un vistazo general de la zona de combate, en la que el jugador obtiene control de su personaje (ubicado en la parte inferior de la imagen), con lo que podrá moverse (dentro de los límites de la arena) y realizar todas las acciones admitidas por la entidad. Inicialmente las únicas entidades presentes serán el jugador y un dummy, contra el cual se podrán ver y practicar los distintos comandos del personaje principal. En la parte superior izquierda podrán verse 3 textos informativos. El primero indica que con la tecla 'F1' se puede iniciar el combate contra un personaje controlado por una inteligencia artificial, también añade que con la tecla 'F4' se puede activar el modo “Debug Render” el cual hará visibles todas las cajas de colisiones. El segundo texto (en verde) muestra la salud del personaje del jugador. Finalmente, el tercero (en rojo) muestra la salud del personaje enemigo, si hay uno activo (en la figura 29 vemos un '-' ya que no hay enemigo activado).


*Fig. 29: Zona de combate.*

## Cámara

La cámara es la que define cuanto del mundo es visible en la ventana, tiene varios tipos y se le pueden definir transformaciones en el tiempo. En el prototipo se pueden encontrar 3 tipos de cámara:

- Libre: cámara fija en unas coordenadas y que tiene dimensiones fijas, utilizada en el menú y puede tener aplicaciones para, por ejemplo, “cinematics” (momentos utilizados para desarrollar o contar parte de la historia).
- Enfocada: cámara con dimensiones fijas que está enfocada y sigue a una entidad indicada, se le puede especificar un offset distinto dependiendo de la dirección a la que se mueva su objetivo.
- Batalla: cámara dinámica centrada en el punto intermedio entre las 2 entidades que se estén enfrentando. Sus dimensiones son las mínimas necesarias para poder encapsular los sprites (o las texturas del frame actual) de ambos combatientes, mientras se mantiene la relación de aspecto definida por las dimensiones originales de la ventana.


*Fig. 30: Jugador posicionado cerca del dummy.*

Podemos comparar las figuras 29 y 30 para ver las diferencias presentadas por la cámara de batalla (que toma de referencias al personaje del jugador y el dummy).

## Recolector de Datos

Para poder obtener datos sobre los combates se decidió crear una funcionalidad para construir un fichero con la información de interés. Cada vez que un combate entre el jugador y una IA acaba (es decir, uno de los 2 pierde toda su salud) se guardan los datos en formato numérico en el siguiente orden:

*[AIType]* : Definido en una enumeración con el mismo nombre, sus valores son del 0 al 2, representando *Basic*, *Scripted* y *Learning* respectivamente.
*[Intentos]* : Cantidad de batallas totales realizadas contra esa IA, por ejemplo si es la primera batalla habrá un 1.
*[Tiempo de batalla]* : Tiempo en mili-segundos que duró el combate.
*[¿Gana el jugador?]* : Booleano que indica si esta batalla ha sido ganada por el jugador.
*[Salud restante]* : Cantidad de salud que le ha quedado al ganador de la batalla.

Utilizando este formato un fragmento de ejemplo del fichero “Results” que se genera podría ser de la siguiente manera:

*0*
*4*
*91296*
*1*
*10*
*0*
*5*
*106720*
*0*
*90*

*0*
*6*
*125184*
*1*
*30*

Aquí se representan 3 combates, ya que se guardan 5 datos por combate (y hay 15 en el ejemplo). De las primeras 5 líneas podemos ver que:

*0* => Se refiere a la IA básica: *Basic*
*4* => Es la 4º pelea que tiene contra esta IA.
*91296* => El combate ha durado 91.296 segundos, poco más de 1 minuto y medio.
*1* => [True] El ganador ha sido el jugador.
10 => Al acabar el combate solo le quedaban 10 puntos de salud.

También se puede comentar que al iniciar el juego se verifica la existencia de este fichero y se actualiza la cantidad de intentos sobre cada tipo de inteligencias artificiales, de modo que la información se mantenga consistente en caso de haber cerrado y re-abierto la aplicación durante las pruebas.

# Extensiones

A continuación se comentan algunas ideas de posibles extensiones que podrían realizarse en el prototipo de cara a acercarlo más a un modelo de juego acabado.

## Terreno de Combate

Actualmente el terreno de combate es limpio y simple, por lo que una posible idea sería añadir terrainboxes (acompañadas de las ayudas visuales pertinentes) en el terreno donde se lleva a cabo la batalla. Estas cajas podrían implementar funcionalidades de obstáculos (para bloquear ataques o burlar al enemigo), trampas (que puedan aplicar efectos negativos o directamente dañar a uno o ambos participantes) o incluso llevar a los jugadores a otra arena diferente, permitiéndoles viajar y alternar entre zonas. Algunas de estas ideas también implicarían actualizaciones en el sistema de cámaras o de interacciones (por ejemplo, que un jugador pueda ser aturdido al ser lanzado contra una columna).

## Personajes

Otra posibilidad sería la de crear más personajes con distintas características y movimientos, de modo que cada uno pueda plantear distintas variantes de estilos de juegos, como por ejemplo, tener un personaje que se mueva lentamente, pero tenga más salud, reciba menos daño y tenga ataques lentos y amplios.

### Acciones

Otra manera de extender el juego actual sería la de trabajar y pulir más el personaje del jugador, añadiendo más movimientos y diversidad de ataques.

### Modo Historia

Finalmente, una manera de poder mejorar este prototipo y acercarlo al concepto de juego final sería la de implementar las funcionalidades básicas del modo historia. Se pueden aprovechar algunas de las funcionalidades actuales (como la cámara, los movimientos o las colisiones), así como implementar otros elementos y funcionalidades necesarias (como por ejemplo, textos animados, conversaciones y un sistema de misiones).

## Posibles Cambios

Actualmente el prototipo tiene varias características que podrían cambiarse o mejorarse, ya sea de cara al rendimiento o para acercarse más al concepto del sistema en las ideas originales. Se podría revisar el sistema de animaciones y comandos para mejorar el rendimiento a largo plazo del juego. También se podría mejorar el sistema de verificación de colisiones para evitar realizar comparaciones entre elementos demasiado alejados uno del otro. La colisión de terrainboxes entre entidades también podría añadirse, para evitar que éstas se atraviesen como fantasmas, influenciando en la jugabilidad ya que puede limitar las vías de escape, así como facilitar el acertar con algunos tipos de ataques.

# 4. La Inteligencia Artificial

## 4.1 Ideas y Conceptos

El sistema de Inteligencia Artificial es la segunda gran parte de este proyecto, en la que se busca poder construir y analizar una variedad de versiones que incluyeran o combinaran distintas técnicas para definir el comportamiento de los personajes no jugadores. La idea principal es que cada entidad, que lo necesite, tenga definida su propia inteligencia con las especificaciones que se adecuen a ella.

El primer método, y el más sencillo de pensar, es el scripting, en el que el programador predefine cómo reaccionará la entidad ante unas determinadas condiciones. Para esto es necesario que la persona considere todos los posibles casos en los que quiera que la entidad reaccione de una manera particular, lo que también presenta el riesgo que haya situaciones no especificadas (por la combinación de condiciones) que puedan causar inacción por parte del personaje, cuando se busca lo contrario. La complejidad de desarrollar este método está dada por las condiciones a tener en cuenta para determinar las situaciones, así como por la variedad de acciones de las que disponga la entidad, ya que es necesario traducir todos los conceptos al lenguaje específico usado para desarrollar el juego.

El segundo método que consideraremos es el aprendizaje automático, el cual consta de una gran variedad de técnicas cuyo objetivo es poder dotar de la capacidad de aprendizaje y "evolución" al sistema o entidad que los utilice. La propuesta para usar el aprendizaje de forma íntegra es poder crear un personaje que aprenda del comportamiento del jugador y lo imite, la razón de esto viene dada más por la idea de la historia y el efecto que se quiere generar en el jugador al "enfrentarlo a sí mismo". En resumen, queremos un personaje que comience sin saber nada y aprenda como se mueve y actúa el jugador para imitarlo de la manera más parecida posible.

El tercer método involucra una mezcla de los 2 anteriormente mencionados, de forma que el personaje disponga de un comportamiento predefinido para ciertas situaciones, pero que también tenga que pasar por un proceso de toma de decisión, de manera tal que pueda modificar algunas características de su comportamiento inicial para actuar en consecuencia a su enemigo o en relación a su entorno.

## 4.2 Implementaciones

Basándonos en las ideas se procedió a realizar la implementación, en la cual se plantearon 3 modelos con distintas cualidades y nivel de complejidad. 2 de ellos se basan en el scripting, mientras que el tercero utiliza aprendizaje automático adaptando la técnica básica de Q-Learning. Por cuestiones de tiempo no se pudo terminar de diseñar y desarrollar una versión que adapte el tercer método mencionado en el apartado 4.1 (la combinación de scripting y aprendizaje). Cada modelo se crea en base a una primera clase genérica *ObjectAI*, que define las funciones básicas para que la IA se actualice y ejecute de manera correcta. Es posible valorar la diferencia en complejidad de las versiones observando las definiciones de las clases, que pueden verse en las figuras 31, 32 y 33.

## Modelo Básico

Este modelo se basa en crear un comportamiento muy básico a través del scripting. Para ello se consideran unas pocas condiciones y un pequeño conjunto de acciones posibles. Lo único que tiene en cuenta la entidad con este comportamiento es la distancia y dirección con su objetivo, de modo que si se encuentra demasiado lejos lo perseguirá; en cambio, si se encuentra lo suficientemente cerca lanzará un ataque al azar con su brazo izquierdo o derecho.

```
class BasicAI :
    public ObjectAI
{
    public:
        BasicAI(WorldObject* owner, sf::Uint32 updateRate = static_cast<sf::Uint32>((1 / 20.f) * IN_MILLISECONDS));
        virtual ~BasicAI();

        virtual void Update(sf::Uint32 diff);
        virtual void Execute() override;
};
```

*Fig. 31: Definición de BasicAI.*

## Modelo con Scripting

Esta versión, como lo indica su nombre, también utiliza scripting, pero se distingue del modelo básico porque considera muchos otros elementos del entorno y tiene un conjunto de acciones más amplio. Utiliza un estilo muy comúnmente visto en combates de jefes, los cuales suelen tener algún tipo de ventajas o poderes más impactantes. Para simular este efecto, esta IA utiliza información sobre, por ejemplo, las animaciones del jugador para poder actuar en consecuencia, como saber si está atacando para protegerse. Otra de las ventajas de las que dispone es que puede utilizar algunas habilidades aunque éstas estén en enfriamiento, es decir, que no pueden usarse hasta que pase un determinado tiempo.

En el comportamiento de esta versión se pueden remarcar varias cualidades, entre las que podemos encontrar una actitud más agresiva si tiene más de un porcentaje de salud. Que intente realizar combinaciones de jabs (doble golpe rápido realizado con un brazo, seguido del otro) y luego toma algo de distancia. En algunas situaciones es capaz de alternar jabs a gran velocidad, lo que puede forzar al jugador a utilizar el Jump Out para escapar.

```
enum MoveType
{
    MOVE_RUN    = 0,
    MOVE_DASH,

    MOVE_NONE
};

enum MoveAction
{
    ACT_CHASE   = 0,
    ACT_ESCAPE,

    ACT_NONE
};

class ScriptingAI :
    public ObjectAI
{
    public:
        ScriptingAI(WorldObject* owner, sf::Uint32 updateRate = static_cast<sf::Uint32>((1 / 20.f) * IN_MILLISECONDS));
        virtual ~ScriptingAI();

        virtual void Update(sf::Uint32 diff);
        virtual void Execute() override;

        virtual void Damaged() override;

    private:
        float m_HealthHitThresh;
        MoveType m_MoveT;
        MoveAction m_MoveA;
        sf::Uint32 m_EscapeCd, m_BlockTimer, m_AttackCd, m_MovingTimer;
        sf::Uint8 m_HitCount, m_MaxHits;
        bool m_Blocking, m_Parrying, m_Attacking;
};
```

*Fig. 32: Definición de ScriptingAI.*

## Modelo con Aprendizaje

Este tercer modelo implementado adapta y utiliza el aprendizaje por refuerzo para poder valorar las acciones a realizar dependiendo del estado de la partida. Sus propias acciones ayudan a modificar la valoración de las mismas para un momento dado, pero la mayor influencia proviene de lo que realiza el jugador mientras combate contra otros enemigos, es decir, que la IA aprende del jugador cuando éste lucha contra cualquiera de los 3 modelos (*Basic*, *Scripting* o *Learning*).

```
const float PLAYER_INFLUENCE_VALUE = 1.f;
const float PLAYER_INFLUENCE_EXTRA = 0.25f;
const sf::Uint8 EPS_GREEDY = 20;

class LearningAI :
    public ObjectAI
{
    public:
        LearningAI(WorldObject* owner, sf::Uint32 updateRate = static_cast<sf::Uint32>((1 / 20.f) * IN_MILLISECONDS));
        virtual ~LearningAI();

        virtual void Update(sf::Uint32 diff);

        virtual void Execute();

        void LearnFromCommand(Command cmd, State& s, bool press);
        /*
        Starting AI commands:
        Up
        Down
        Left
        Right
        Attack 1
        Attack 2
        */
    private:
        std::map<Command, std::pair<std::list<std::pair<State, float>>, std::list<std::pair<State, float>>>> m_States;
        std::list<std::pair<State, float>>::iterator m_StateItr;
        std::vector<bool> m_LearntCommands;
        bool m_UpdateQ;

        float GetNextStatesMaxQ(const State& s);
        void InitBasicStates(HealthState o, HealthState t);
        void InitNoActionStates();
        void GetAvailableOptions(std::list<std::pair<std::pair<Command, bool>, std::pair<float, std::list<std::pair<State, float>>::iterator>>>& options);
};
```

*Fig. 33: Definición de LearningAI.*

## Estado

Definir el estado es un proceso importante para el diseño de la IA, ya que dependiendo de lo que se haga puede cambiar la complejidad y el rendimiento del sistema, así como garantizar el éxito o el fracaso del objetivo. Se decidió por utilizar información de los 2 personajes relacionados en el combate, así como cuantificar estos valores en una cantidad que intentara aportar suficiente información para trabajar, sin que la complejidad se elevara a niveles intratables.

Se han intentado considerar las variables que pudiesen condicionar la toma de decisiones, considerando la salud (de ambos participantes), el estado (de ambos), la distancia entre las 2 entidades, la posición de quien está valorando las opciones respecto a su oponente y, si éste primero está mirando en dirección al segundo. La salud y la distancia están divididas en 3 categorías, el estado es una generalización de la animación que está realizando la respectiva entidad (ignorando la dirección de la misma, por ejemplo: *Idle*), la posición puede ser una de las 8 direcciones posibles y, finalmente, el hecho de estar encarado a la dirección del oponente es un booleano.

```
private:
    TargetDistance m_Dist;
    WorldObjectPosIndex m_OwnerPos;
    HealthState m_OwnerHealth, m_TargetHealth;
    ObjectState m_OwnerState, m_TargetState;
    bool m_OwnFacesTar;
```

*Fig. 34: Variables de la clase del estado: State.*

## Comandos y Aprendizaje

El modelo con aprendizaje comenzará con un registro inicial de comandos y estados para poder tener la capacidad de realizar acciones básicas incluso en su estado inicial. Los comandos con los que comienza son *Up*, *Down*, *Left*, *Right*, *Attack 1* y *Attack 2*, los cuales se utilizan para que tenga la capacidad de moverse y atacar. A partir de este punto el valor de calidad de cada acción dado un estado podrá ir variando en 2 ocasiones:

- Aprendizaje propio: modifica el valor de calidad de alguna acción, dado un estado, según si el estado resultante es valorado de mejor manera.

- Aprendizaje del jugador: modifica el valor de un comando, o registra uno nuevo, dependiendo del input del jugador. Esto solo es aplicable si la acción tiene algún efecto y es realizada durante un combate (jugador contra entidad controlada por alguna IA).

Es necesario especificar que esta IA no puede realizar acciones que no haya aprendido, a parte del conjunto inicial. Por lo que si el jugador no realiza (en contra de cualquier enemigo con IA), por ejemplo, la acción de *Dash*, entonces la entidad con este comportamiento nunca podrá usar ese movimiento. A su vez, para este personaje existen las mismas restricciones que para el jugador, como tiempo de enfriamiento para algunos movimientos, o que no pueda realizar algunas acciones en

determinado estado (por ejemplo, no puede lanzar un golpe mientras se ejecuta el aterrizaje del Jump Out). Por ello cada vez que busca decidir qué acción realizar debe filtrar aquellas que ha aprendido y que puede realizar según su propio estado, una vez tiene ese listado con sus respectivos niveles de calidad llevará a cabo la acción con mayor calidad, aunque también tiene asignada una probabilidad para seleccionar una acción al azar de entre las disponibles.

```
void LearningAI::LearnFromCommand(Command cmd, State& s, bool press)
{
    if (!m_LearntCommands[cmd])
    {
        // If the AI didn't know the command, we "teach" it
        std::list<std::pair<State, float>> t;

        m_LearntCommands[cmd] = true;
        t.push_back(std::make_pair(s, 1.f));

        if (press)
            m_States.insert(std::make_pair(cmd, std::make_pair(t, std::list<std::pair<State, float>>())));
        else
            m_States.insert(std::make_pair(cmd, std::make_pair(std::list<std::pair<State, float>>(), t)));
    }
```

*Fig. 35: Fragmento de la función LearnFromCommand para aprender basándose en una acción del jugador. En el fragmento se ve como se registra un comando nuevo en la IA.*

## 4.3 Posibles Cambios

En este apartado se comentan algunos aspectos de la implementación que podrían mejorarse, o podrían haberse mejorado si se hubiese tenido más tiempo para dedicar al desarrollo de la IA.

### Definición del Estado

Uno de los principales, y más importantes, aspectos se encuentra en la definición del estado. La definición actual no permite generalizar muchas de las acciones y reduce el ritmo del aprendizaje, el cual debería ser mayor para poder corresponderse a las ideas y conceptos originales. Se podrían combinar algunas de las variables para expresar hechos más concretos que aporten más información, a partir de los cuales poder definir las acciones. Los comandos son otro elemento que limita y ralentiza el proceso de aprendizaje, por lo que se podrían especificar acciones particulares a las cuales vincular los estados, un ejemplo de acción podría ser “moverse hacia el enemigo” o “alejarse del enemigo”, la cual se puede generalizar de manera básica a todas las direcciones.

### Valoración de las Opciones

Más que un posible cambio esto sería un experimento, en el que se cambiaría la manera de valorar las opciones disponibles, cambiando la calidad de los estados, así como la posibilidad de realizar una acción al azar, en lugar de la mejor calificada. La idea es poder ver el impacto que esto pueda tener en la calidad y velocidad del aprendizaje de la entidad, si se acerca o aleja del objetivo a mayor o menor ritmo.

## Movimientos Únicos

Este cambio, o extensión, está orientado a las IAs que utilicen scripting, el cual busca dotarles de otras características únicas no accesibles por el jugador. En este caso se trata de técnicas o movimientos únicos, ya sean ofensivos o defensivos, que permitan añadir complejidad y personalidad a la entidad. Esto es usado en casi todos los jefes en los videojuegos para definir mecánicas del encuentro y que son, o generalmente intentan serlo, únicas, tanto a nivel de idea como en el apartado visual.

# 5. Resultados

## 5.1 Análisis de Datos

Todos los datos de las batallas realizadas por los testers se ha recopilado en un fichero para poder extraerlos y procesarlos de manera más sencilla. A continuación, se visualizarán algunos de los datos obtenidos a lo largo de varias pruebas realizadas por distintas personas, es necesario tener en cuenta que como la selección de la IA contra la cual se lucharía era aleatoria la cantidad total de intentos realizada contra cada una puede variar.

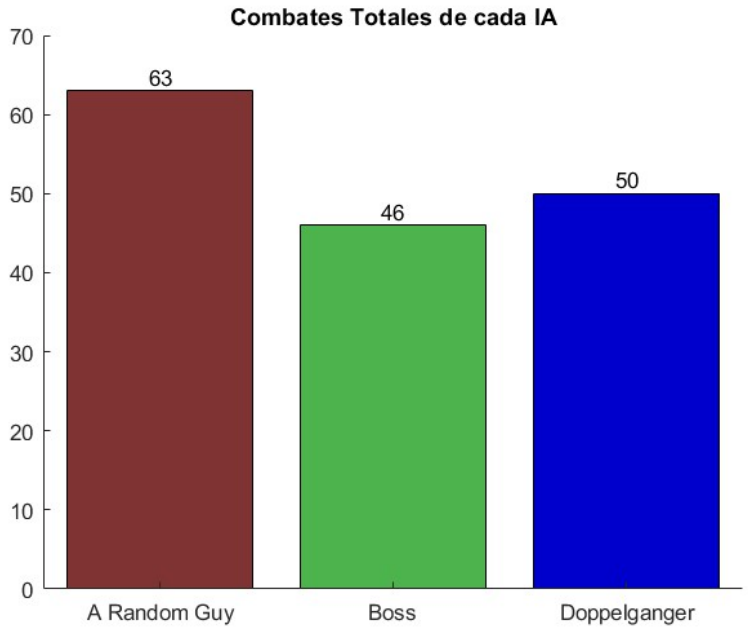



*Fig. 36: Comparación de la cantidad de combates en total realizados por cada implementación.*

En la figura 36 podemos ver un gráfico de barras que nos muestra la cantidad total de combates que se ha llevado a cabo contra cada sistema de IA implementado:

- A Random Guy: 63
- Boss: 46
- Doppelgänger: 50

Lo que nos deja con un total de 159 combates realizados entre los jugadores y las 3 IAs implementadas. A partir de este punto extraeremos datos de modo particular para mostrarlos y compararlos entre sistemas. En la figura 37 se puede observar la distribución de cada versión dado un momento (identificado por el número de aparición), es posible ver que "A Random Guy" es la única que ha aparecido más de 15 veces durante las pruebas, mientras que "Doppelgänger" solo ha llegado en una de las sesiones de prueba a pasar de 9 apariciones.

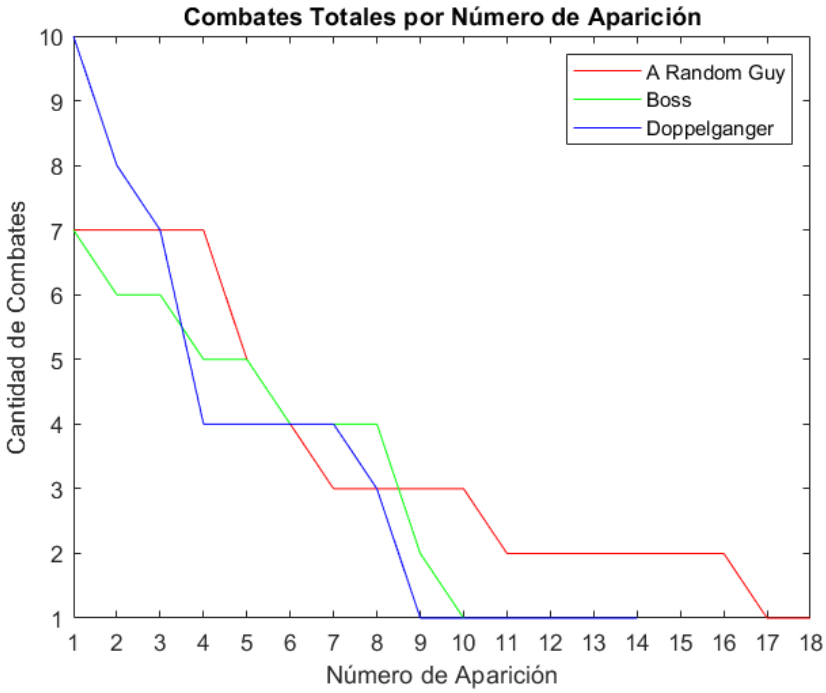



*Fig. 31: Cantidad de apariciones de las versiones.*

Lo siguiente a analizar será la cantidad de victorias que cada una de las inteligencias ha tenido en relación a su número de aparición. La gráfica puede verse a continuación en la figura 32.


*Fig. 32: Victorias de cada IA según su número de aparición.*

De esta gráfica podemos observar como "A Random Guy" ha ganado todos los combates de sus apariciones 1 y 3, con una derrota en la 2. Por otro lado "Boss" ha ganado 4 de 7, y "Doppelgänger" solo 3 de 10. Las primeras apariciones son un punto particular ya

que, en general, los jugadores no conocen como se controla el juego, ni todos sus movimientos, por lo que las versiones con scripting son capaces de explotar ese desconocimiento, mientras que la que utiliza aprendizaje deja mucho más margen al jugador para responder, ya que ella misma tiene conocimientos y valores básicos, es como una pelea entre 2 personajes que no saben como jugar exactamente. Cuanto mayor el número de apariciones, más partidas ha jugado la persona, por lo que ella va aprendiendo a como moverse y responder, así como los comportamientos que tienen, particularmente, "A Random Guy" y "Boss". A pesar de que también recibe conocimientos, "Doppelgänger" debe enfrentar a jugadores humanos que tienen un ritmo de aprendizaje mayor, con lo que se puede ver claramente que no consigue superarlos. Algunas de las rachas de victoria continúan viéndose, principalmente, en la versión básica, ya que tiene un comportamiento muy agresivo que deja poco espacio al jugador para pensar y reaccionar, es una IA que fuerza a la persona a jugar de forma reactiva. Algo que "Boss" genera en menor medida, dando más espacio al jugador para poder probar el estilo que se ajuste más a sí mismo, dentro de los límites de los controles que posee el prototipo.

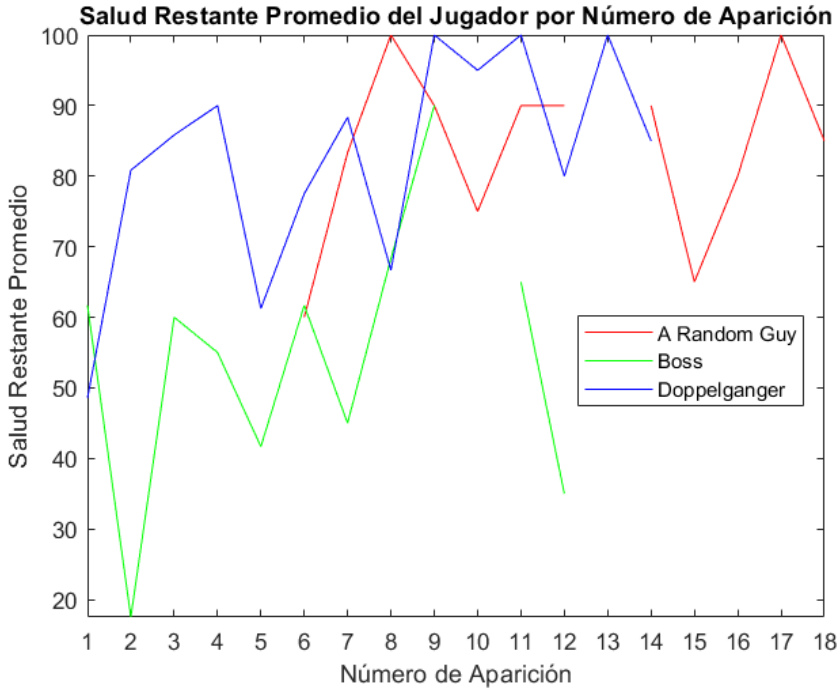



*Fig. 33: Salud restante del jugador en combates ganados.*

En la figura 33 se puede ver una gráfica que indica cuanta salud le ha quedado al jugador al ganar los combates contra cada IAs, según el número de aparición de ésta. Se pueden ver zonas discontinuas, que son las marcadas por las apariciones en las que la IA correspondiente ganó todas las peleas, por ejemplo el comienzo de "A Random Guy" está plenamente dominado por él. En general aquí se puede notar también el ritmo de aprendizaje del jugador, ya que se nota como cada vez ha conseguido victorias con mayor salud, lo que implica que recibió menos daño de la IA. Para ello se necesario aprender el comportamiento de las versiones con scripting, sabiendo como y cuando atacan, para defenderse o escapar. Por otro lado, se puede ver como "Doppelgänger" llega a un punto en el que poco a poco alcanza menos al jugador, pero en otras apariciones posteriores logra volver a empezar a tocarlo, lo que muestra algo de evolución por su parte, aunque particularmente lenta.

Los resultados de las pruebas nos permiten ver que los sistemas implementados pueden presentar diversos desafíos, pero que aún necesitan ser trabajados y pulidos. Los modelos con scripting cuentan con ventajas de reacción inhumanas, que les permiten poder volverse dominantes del combate con facilidad (como pasa con "A Random Guy" y su conducta de solo perseguir y atacar). Por su parte, el modelo con aprendizaje necesita de un mejor diseño, principalmente en el modelado del estado de forma que se permita agilizar la velocidad de evolución, lo que, en este caso, implica una mejor imitación del jugador.

## 5.2 Valoraciones del Testing

Una vez terminadas las pruebas del prototipo, los testers procedieron a responder una pequeña encuesta, la cual nos permitirá ver cuales son sus valoraciones y opiniones respecto al sistema que han probado y la experiencia vivida. La mayoría de las preguntas están compuestas por un nivel de granularidad que cataloga valoraciones en un rango entre 1 y 5. También se ha dejado un espacio libre para que pudieran transmitir su opinión de forma íntegra, de la cual se buscará sacar algunos datos respecto al prototipo visto desde un punto de vista diferente.

Valora la jugabilidad del prototipo

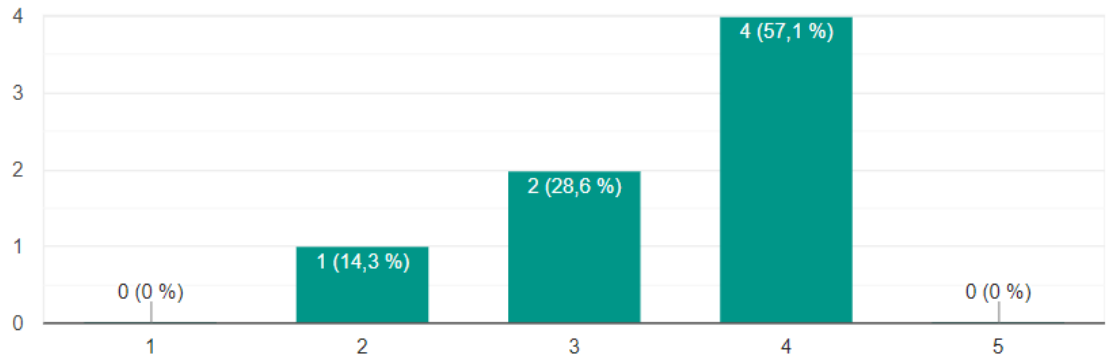



*Fig. 34: Gráfico de barras de la valoración.*

La figura 34 muestra la valoración de los testers en una escala de 1 (Muy mala) a 5 (Muy buena), en la que se puede ver fácilmente como la mayoría ha valorado de forma positiva las mecánicas de juego y movimientos planteados en el prototipo.

¿Qué enemigo de ha parecido más desafiante/difícil?

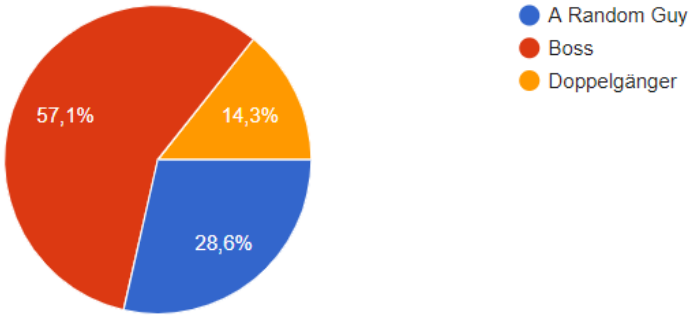



*Fig. 35: Gráfico circular de la elección.*

En la figura 35 podemos ver la distribución de quien ha sido considerado el enemigo más desafiante. El resultado está dentro de lo esperado por el tipo de comportamiento más impactante y dinámico que presenta "Boss", mientras que "A Random Guy" puede ser difícil hasta que logras medir los ataques que lanza de forma predecible. En cierta manera hay que valorar que "Doppelgänger" haya sido considerado como el desafiante o difícil por uno de los testers, más que nada porque tiene un ritmo de aprendizaje lento y esto es algo que lo puede dejar a la sombra cuando comparamos el rendimiento final de todas las versiones.

¿Cuán satisfactorio ha sido el combate contra "A Random Guy"?

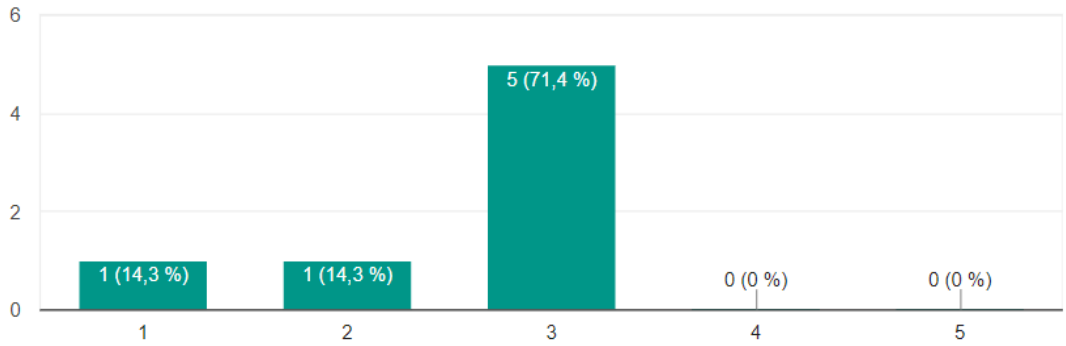



*Fig. 36: Distribución de la satisfacción de "A Random Guy".*

En la figura 36 se puede ver como las valoraciones de satisfacción de la implementación más básica no llega a tener una muy buena calificación, quedando en un punto medio entre lo neutro y la insatisfacción. Considerando el tipo de comportamiento es de esperar que a la larga los jugadores sientan que es un enemigo simple y molesto, por perseguir y atacar sin parar. No deja espacio para que el jugador pueda elegir como actuar, solo le deja reaccionar y tomar medidas rápidas y simples.

¿Cuán satisfactorio ha sido el combate contra “Boss”?

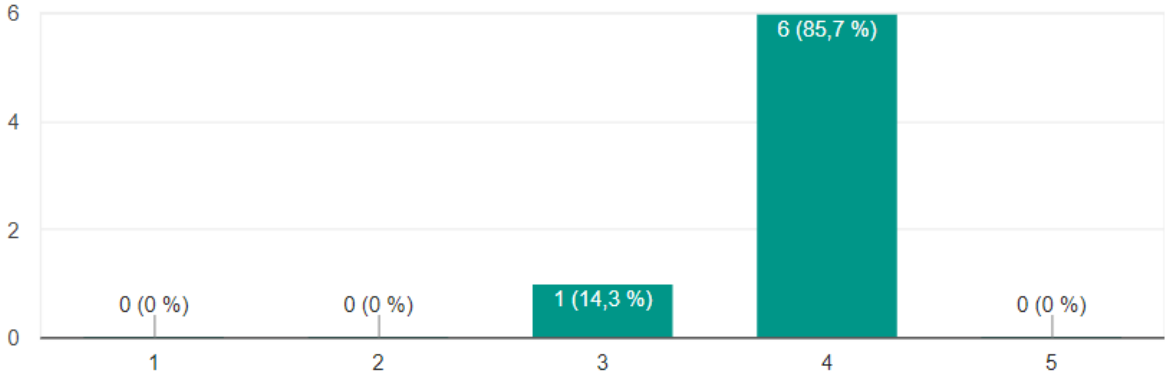



*Fig. 37: Valoración de satisfacción de "Boss".*

Podemos observar, en la figura 37, que “Boss” ha sido un combate más disfrutado por los testers, ya que, a pesar de estar pre-definido, deja un espacio al jugador para reaccionar, además de usar movimientos más agradables visualmente, que incluso pueden dar una pista al jugador de lo que puede ser capaz de usar (aunque no con las mismas limitaciones). Se pude concluir que esta implementación ha sido satisfactoria para la gran mayoría, aunque puede mejorarse.

¿Cuán satisfactorio ha sido el combate contra “Doppelgänger”?

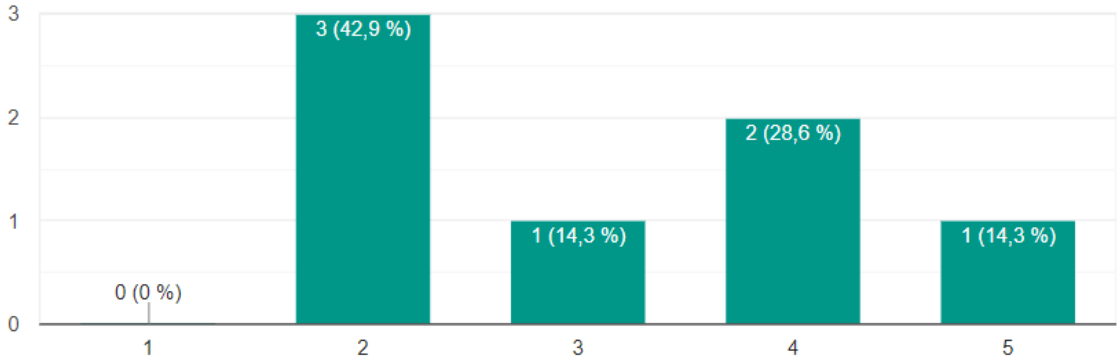



*Fig. 38: Valoración de satisfacción de "Doppelgänger".*

En la figura 38 podemos observar lo dispersas que son las valoraciones de la implementación con aprendizaje automático, aunque se pueden redondear en algo entre lo neutral y lo satisfactorio. “Doppelgänger” era, sin duda, el encuentro más particular de los 3, ya que el comportamiento y los movimientos estaban completamente ligados a lo que hiciera el jugador durante ese combate o contra las otras IAs. Como con “Boss”, se puede sacar una valoración positiva de la actual implementación, pero aún tiene mucho espacio por el cual mejorarla.

¿Qué nivel de cambio has notado en “Doppelgänger” a lo largo de las pruebas?

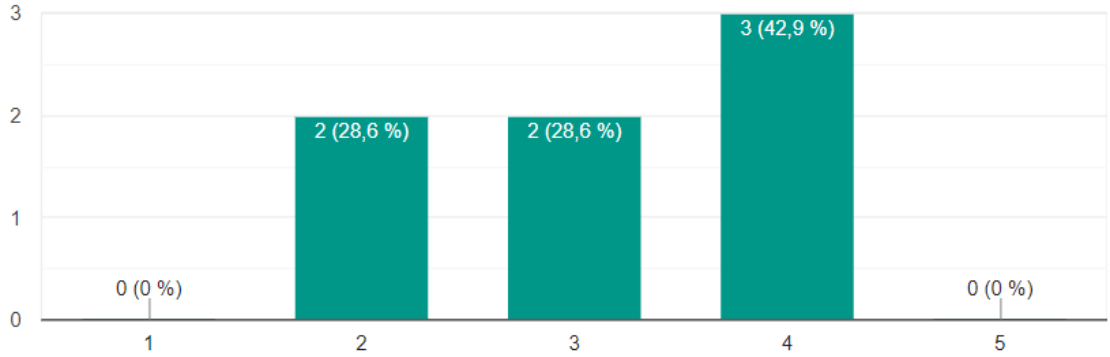



*Fig. 39: Percepción del cambio de comportamiento de "Doppelgänger" por parte de los usuarios.*

Además de las pruebas estadísticas que nos permitan comprobar la evolución en el comportamiento de la implementación con aprendizaje, es importante tener una valoración de lo que perciben los usuarios. Esta pregunta existe con esa finalidad, y es en las respuestas que podemos ver que el cambio puede ser percibido, aunque siendo en menor o mayor medida según las circunstancias. Lo ideal habría sido poder ver un cambio sustancial hasta un determinado punto, pero para una primera versión de prototipo estos resultados, a nivel de percepción de cambios, no son desalentadores.

“Doppelgänger” es una IA que intenta aprender del comportamiento del jugador para imitarlo ¿Cuánto consideras que “Doppelgänger” se asemejaba al estilo de juego que usabas en el prototipo durante el último combate que has tenido contra él?

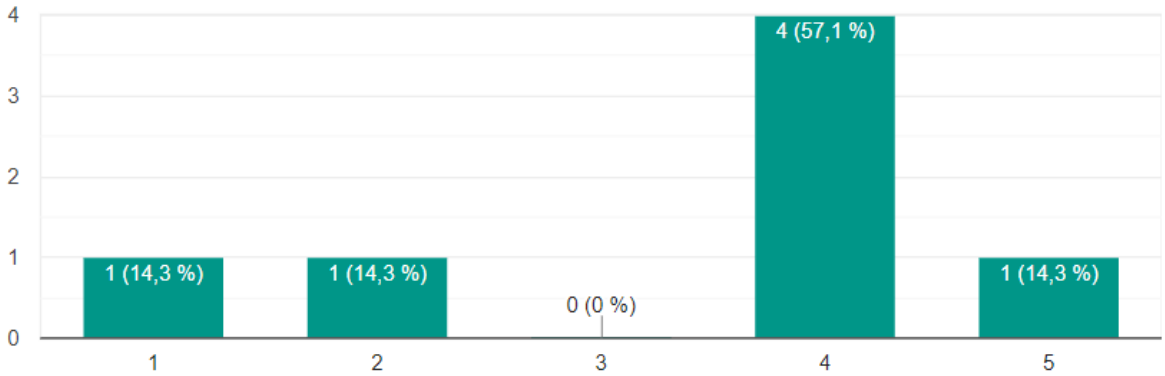



*Fig. 40: Valoración de semejanza en estilo de juego.*

A parte de considerar solo el cambio en el comportamiento se ha considerado necesario tener en cuenta una valoración de la semejanza, ya que era el objetivo principal de la IA con aprendizaje automático. Viendo los resultados mostrados en la figura 40 podemos considerar que, desde el punto de vista de los usuarios, se ha podido cumplir hasta cierto punto con el objetivo. La mayoría ha considerado que los movimientos y la conducta eran bastante semejantes (en un caso hasta casi idéntico), pero también hay que tener en cuenta lo como cambia lo que se percibe de persona a persona, ya que hay 2 testers que han valorado negativamente el nivel de semejanza. Con esto se puede concluir, como antes, que vamos por un buen camino, pero que aún hay bastante por solventar y mejorar.

Valora cuanto consideras que puede aportar el uso de aprendizaje automático en la IA de los videojuegos

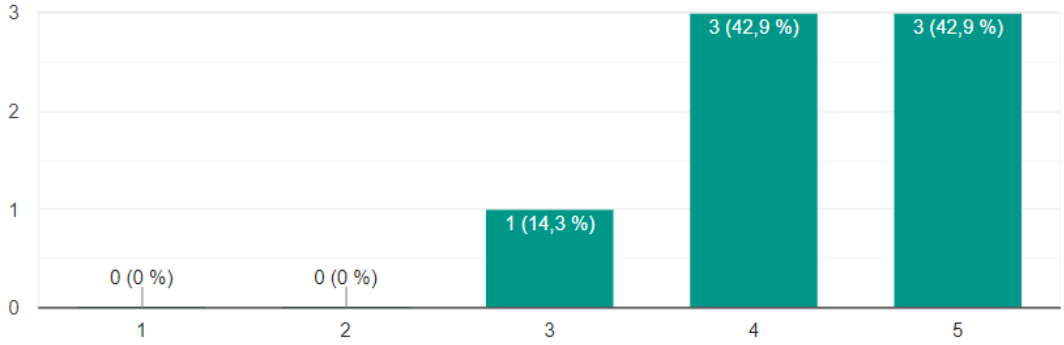



*Fig. 41: Valoración del aporte del aprendizaje automático a los videojuegos.*

Hemos considerado como un buen dato de interés cuanto creen los testers que este tipo de métodos y técnicas pueden aportar en la cuestión del desarrollo de IAs para videojuegos. La gran mayoría ha expresado una confianza en que el aprendizaje automático pudiese aportar nuevas ventajas y propuestas interesantes para este mundo que experimenta con variedad de elementos.

¿Qué opinión o valoración tienes sobre el juego/prototipo?

La última parte de la encuesta dejó un espacio de opinión abierto para que los testers pudiesen señalar y valorar todo lo que consideraran necesario. De entre los comentarios se pueden extraer algunos datos de interés sobre el prototipo, así como la valoración de las ideas y conceptos implementados.

Lo primero a resaltar es en el apartado técnico y es que el prototipo posee un fallo de memoria importante causado por alguna razón desconocida, ya que en algunos sistemas es mucho más impactante que en otros en corto plazo, mientras que a largo plazo parece afectar a todos por igual. Esto es un error importante a reparar de cara a hacer evolucionar el trabajo de un prototipo a una versión jugable. En lo que respecta a la jugabilidad todos la disfrutaron, la notaron dinámica y entretenida. Algunos testers expresaron unos puntos en contra, como que a veces el movimiento no se sentía del todo correcto o que las colisiones necesitaban ser más exactas. También hubo algunos comentarios remarcando las ventajas de las inteligencias con scripting por sobre la  de aprendizaje, ya sea por el comportamiento pre-definido y agresivo, así como por las ventajas en técnicas y movimientos (el caso del "Boss"). Una observación particular fue la de uno de los testers, que a partir del comportamiento reflejado en "Doppelgänger" se dio cuenta de como estaba condicionado a luchar de manera reactiva por las otras IAs, con lo que decidió tomar más iniciativa en esos combates, llevando a que "Doppelgänger" también sufriera un cambio, lento, en su comportamiento.

# 6. Conclusión

En conclusión, en este trabajo hemos logrado idear, diseñar y desarrollar un prototipo jugable de nuestro videojuego Action RPG 2D. A pesar de tener fallos, uno de bastante relevancia, se pudo comprobar que las ideas y conceptos relacionados a la jugabilidad realmente cumplen con sus objetivos respecto a velocidad, dinamismo y entretenimiento.

Por el lado del estudio de viabilidad de inteligencia artificial con aprendizaje automático, podemos concluir que es algo muy condicionado por lo que se busque reproducir, así como por los recursos y especificaciones de los sistemas para lo que se la está pensando. Para nuestro caso en particular es posible aplicarla y todavía hay espacio a mejorar muchos de sus aspectos, aplicando otras técnicas u otras decisiones de diseño.

Podemos asegurar que aplicar técnicas de aprendizaje automático para mejorar la experiencia del jugador es una idea que llama la atención del público y que, realizado de la manera correcta, puede no solo mejorar la calidad del producto final, sino que también puede ser un poderoso elemento de publicidad para el proyecto.

# 7. Trabajo Futuro

Puede haber 2 caminos para extenderse en el trabajo futuro de este trabajo, uno es el desarrollo y mejora del motor del juego, el otro es en el diseño, implementación y prueba de otros métodos de aprendizaje automático para aplicar en este tipo de videojuegos.

En el primer caso se pueden averiguar y valorar los fallos del sistema actual para poder corregirlos, también se puede comenzar a plantear el desarrollo de algunas de las extensiones mencionadas anteriormente (en el apartado “Extensiones” de la sección 3.2). Otro ámbito es el de priorizar una implementación eficiente, así como extensible y/o generalizada de las distintas partes.

En el segundo caso se pueden realizar una mayor variedad de experimentos con variedad de versiones, probando distintas alternativas para poder evaluar los cambios de eficacia y eficiencia. Una posibilidad es aplicar cambios en los parámetros o elementos del diseño en la implementación actual, así como realizar  análisis de viabilidad y pruebas utilizando otras técnicas y algoritmos de aprendizaje automático.

Finalmente se debería diseñar, implementar y realizar las pruebas pertinentes de un sistema que intente mezclar de forma adecuada el scripting y el aprendizaje, de modo que se pueda crear un comportamiento base el cual pueda aprender y adaptarse con el fin de ofrecer una experiencia mejor y más dinámica. La clave es conseguir crear un sistema con capacidad adaptativa y consistente, de modo que no tenga fallos en el comportamiento, así como que sea un desafío posible de superar.

# 8. Gestión del Proyecto

## 8.1 Organización y Planificación

### Metodologías

Con el objetivo de acabar de manera exitosa el proyecto se planteará una práctica de desarrollo iterativo e incremental, así como una metodología de desarrollo en espiral. La idea es trabajar en pequeñas partes y funcionalidades, realizar varias iteraciones sobre cada una hasta conseguir una versión satisfactoria. Luego se continuará con otra parte siguiendo el mismo método, de modo que poco a poco, de modo incremental, se vayan completando los fragmentos del proyecto. Acompañando este tipo de planificación irán las definiciones de fechas límites para las distintas funcionalidades en las que se van trabajando.

### Herramientas

Para la implementación del proyecto se utilizará el lenguaje C++ y utilizará el entorno de desarrollo Microsoft Visual Studio 2017. Para desarrollar el juego se usará SFML (Simple and Fast Multimedia Library) [12], la cual provee una interfaz para facilitar el desarrollo de juegos y aplicaciones multimedia. También se considerará la posibilidad de utilizar mlpack [13], una biblioteca escrita en C++ que provee algoritmos ya implementados de aprendizaje automático.

Para el control de versiones se utilizará un repositorio de Bitbucket [14], de modo que puedan verse fácilmente las modificaciones realizadas en el código, así como el progreso general del proyecto a través de las distintas funcionalidades extras que proporciona la web (como Issues para hacer seguimiento de problemas identificados).

### Posibles Obstáculos

Durante la realización del proyecto pueden surgir contratiempos u obstáculos que ralenticen el progreso. A continuación los listaremos, analizaremos brevemente y comentaremos posibles soluciones.

Errores de Código: es un tipo de error bastante común y que, por sí solo, no causa mucha pérdida de tiempo. La cuestión es que en general estos errores se suelen presentar en grandes cantidades a lo largo de un proyecto lo que acaba causando una pérdida de tiempo que no es negligible. La solución es simplemente intentar evitar estos errores siendo consciente de lo que se escribe, ya que suelen ser causados por despistes en el proceso de escritura el código.

Errores de Lógica: estos errores pueden aparecer con más normalidad en proyectos donde muchas funcionalidades necesitan trabajar en conjunto. Estos fallos pueden consumir mucho tiempo en corregir, ya que a veces pasan desapercibidos, lo que puede causar que se implementen otras funcionalidades sobre una base con un fallo de diseño, causando un gran

desperdicio de tiempo al tener que replantear todo lo que se haya hecho. Es mejor evitarlos antes que solucionarlos, intentando pensar y asegurándose que el sistema que se plantea es correcto.

Entendimiento e Implementación de los algoritmos de IA: la complejidad de estos temas puede convertirse en un obstáculo que provoque una gran pérdida de tiempo. Si se pierde mucho tiempo con este apartado es recomendable acudir al director del proyecto por algo de guía.

Tiempo y Calendario: el tiempo para la realización del proyecto es bastante limitado, por lo que es necesario establecer fechas límites para las distintas funcionalidades intentando organizar el tiempo de la mejor manera posible.

## Validación

La metodología para validar el progreso realizado en el proyecto se realizará, en primera instancia, a través de las pruebas realizadas por el Desarrollador, el Diseñador/Artista y los posibles Beta-testers. El cumplimiento de los objetivos y fechas se podrá ir verificando con la ayuda de Bitbucket [14] y de reuniones realizadas con los distintos actores que participan del desarrollo, ya sean con el director y/o el diseñador.

## Programa

La duración aproximada del proyecto es de 4 meses, comenzando el 19 de Febrero de 2018 hasta las fechas entre el 25 y el 29 de Junio de 2018. La entrega de la memoria se debe realizar una semana antes de la defensa. La distribución de horas y duración de las tareas son una estimación, por lo que se debe tener en cuenta que pueden surgir imprevistos o cambios que modifiquen la planificación planteada a continuación. La idea es poder distribuir el tiempo dependiendo de la complejidad de cada tarea de modo que se pueda finalizar el trabajo dentro del plazo previsto.

## Descripción de las Tareas

### Hito Inicial

Es la primera fase y se enfoca en comenzar la planificación del proyecto, así como la redacción de la memoria. En este punto se definen y desarrollan varios aspectos importantes:

1) El alcance, la contextualización y la bibliografía.
2) La planificación temporal
3) La gestión económica, la sostenibilidad y el compromiso social
4) Presentación preliminar
5) Documento final

## Desarrollo del Juego

El objetivo de esta fase es el desarrollo del motor del juego y todas sus funcionalidades necesarias. El proceso a seguir será según la metodología indicada, por lo que lo primero será definir el objetivo a cumplir en el ciclo, valorar los riesgos que pueda surgir y las posibles maneras de solucionarlos, implementar y verificar el funcionamiento de la parte correspondiente, y, finalmente, planificar la siguiente iteración. Es necesario que esta parte se realice antes de la implementación de la de Inteligencia Artificial, pero puede llevarse a cabo en paralelo con las fases de aprendizaje y diseño de la misma.

A grandes rasgos se pueden identificar varios aspectos que deben trabajarse en esta fase, éstos se listan a continuación.

### Estructura Básica del Juego

Es el código básico que permite el funcionamiento del juego. Esta parte se encargará de iniciar la aplicación y de ejecutar el bucle principal, en el cual se realizan acciones como la actualización del estado de las entidades, el dibujado de las mismas en pantalla y la gestión de entradas registradas por el sistema, por ejemplo: cuando el jugador presiona una tecla. Esta parte debe desarrollarse antes que las demás, ya que es lo que permitirá realizar las pruebas y verificaciones de todas las funcionalidades.

### Animaciones

Es el sistema para poder animar las imágenes, es decir, que se vayan dibujando los distintos fotogramas correspondientes por segundo. Es una parte importante del juego porque sirve como medio para comunicar información al jugador de manera visual, por ejemplo cuando un enemigo ataca. También permite una introducción a un mundo más “vivo”, mejorando la inmersión y la experiencia. Su implementación se puede llevar a cabo con la del prototipo básico para las entidades.

### Entidades

Representación de los personajes y otros objetos dentro del juego. Se componen de muchos atributos y elementos, además de que serán capaces de realizar diversas acciones, como moverse, atacar o esquivar. Las entidades son clave en el juego, representarán al jugador, los enemigos y algunos posibles elementos interactuables del entorno; también podrán permitir la extracción de información de utilidad que pueden permitir valorar de mejor manera los resultados del proyecto.

### Controles

Si bien en la estructura básica del juego se habla de procesar los datos de entrada del jugador, es necesario marcar todo el sistema que soporta esa funcionalidad en su propia categoría. Los controles son otra parte relevante ya que permitirán al jugador interactuar con el entorno, ya sea moviéndose o enfrentando a los enemigos. Registrar esta información es de vital importancia para el sistema de inteligencia artificial. Se debe comenzar a trabajar en esta parte una vez las entidades estén mayormente terminadas, ya que se necesita de ellas para que realicen acciones y se pueda verificar su funcionamiento.

Colisiones

El sistema de colisiones permitirá la interacción entre entidades, o con elementos del entorno, según el estado en que se encuentren. Es otra parte relevante y necesitará que las entidades y animaciones estén terminadas, antes de poder comenzarse a implementar.

Textos

Es un sistema de texto dinámico que se presentará en la pantalla para poder dar contexto al jugador, mostrar interacción entre personajes o progresar con la historia. No tiene una gran relevancia en relación a los objetivos del proyecto, pero sí respecto al juego. No tiene dependencias particulares, excepto por la del bucle principal, por lo que se puede ir trabajando en este apartado poco a poco a lo largo del desarrollo del motor del juego.

Todas estas funcionalidades conforman el "núcleo", una vez terminadas las más relevantes se puede comenzar a trabajar activamente en el sistema de IA.

## Desarrollo de la Inteligencia Artificial

Estudio y Aprendizaje

Esta fase se compondrá de una primera parte aprendizaje, en la cual se dedicarán horas a estudiar e investigar los algoritmos y metodologías que se aplicarán en el proyecto. Esto es necesario principalmente porque se necesitarán conocimientos específicos de inteligencia artificial, en este caso aprendizaje automático. Es una parte crucial, ya que sería imposible llevar a cabo el proyecto sin entender del tema. No hay dependencias en relación a la implementación del juego, pero se esperará a acabar esa parte antes de comenzar con esta, de modo que se pueda asegurar una correcta organización y cumplimiento de los objetivos.

Diseño e Implementación

Una vez el conocimiento sobre el campo del aprendizaje automático, así como el de los algoritmos y técnicas necesarias, sean lo suficientemente amplios se procederá con el diseño del sistema, para su posterior implementación. Se pensará en la información necesaria para el funcionamiento, y la manera en que ésta se transmitirá y procesará. Esto implica la posibilidad de tener que realizar algunos cambios o implementar métodos específicos en el motor del juego. Una vez establecidos los parámetros y el funcionamiento deseado se comenzará a desarrollar el código del sistema, siguiendo la metodología se irán definiendo objetivos a cubrir dentro de la iteración correspondiente, se valorarán los riesgos que pueden surgir y se procederá con la implementación y las pruebas. Esta fase puede requerir de varias iteraciones que irán cubriendo distintos aspectos de la inteligencia artificial.

Pruebas y Verificación

Como se comentó anteriormente, en cada iteración se deberán realizar pruebas para verificar el cumplimiento de los objetivos planteados. Para ello es necesario tener claro los

cambios en el comportamiento que se buscan ver, así como los elementos que puedan causarlos, y comprobar si ocurren. Se deben realizar varios tipos de pruebas, variando la información, antes de poder indicar si una funcionalidad es correcta o necesita ser corregida.

### Recolección y Evaluación de Datos

Una vez se tenga un prototipo se comenzarán a realizar algunas pruebas para recolectar datos del funcionamiento en un estado temprano, con cada versión mejorada y aceptable se procederán de la misma forma. Se procesará información acerca de la interacción entre el jugador y los enemigos, enfocándose en aspectos como el tiempo de enfrentamiento, el daño recibido o las derrotas sufridas; también se pedirá la opinión y valoración de los testers que realicen las pruebas. Todo esto permitirá la formulación de una conclusión final respecto al cumplimiento de los objetivos finales del proyecto.

### Hito Final

En esta última parte se procederá a redactar y presentar la memoria del proyecto, también se entregará el código fuente y una versión compilada del juego desarrollado. A la documentación se le añadirán anexos que aporten información acerca del propio juego y su funcionamiento, así como del proceso creativo y técnico detrás de su implementación. El proyecto se cerrará una vez se acabe y entregue la memoria en la fecha correspondiente, una semana después de hacerlo se deberá realizar la presentación y defensa del proyecto.

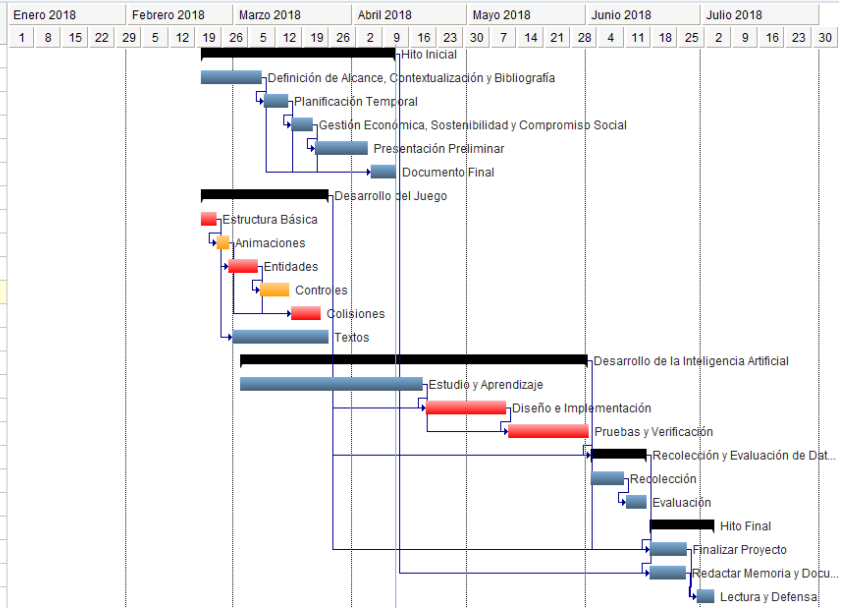
## Duración aproximada

| **Tarea** | **Duración Aproximada en Horas** |
|---|---|
| Hito Inicial | 90 |
| Estructura Básica del Juego | 8 |
| Animaciones | 8 |
| Entidades | 34 |
| Controles | 16 |
| Colisiones | 24 |
| Textos | 10 |
| Estudio y Aprendizaje de IA | 140 |
| Diseño e Implementación | 85 |
| Pruebas y Verificación | 80 |
| Recolección y Evaluación de Datos | 20 |
| Hito Final | 40 |
| **Total** | **555** |

## Diagrama de Gantt

Project Name **Desarrollo de Inteligencia Artificial para Videojuego Action RPG 2D**

| | Nombre | Inicio | Fin | Asignado a |
|---|---|---|---|---|
| 1 | **Hito Inicial** | **19/02/2018** | **09/04/2018** | |
| 2 | Definición de Alcance, Contextualización y Bibliografía | 19/02/2018 | 06/03/2018 | Desarrollador |
| 3 | Planificación Temporal | 07/03/2018 | 13/03/2018 | Desarrollador |
| 4 | Gestión Económica, Sostenibilidad y Compromiso S... | 14/03/2018 | 19/03/2018 | Desarrollador |
| 5 | Presentación Preliminar | 20/03/2018 | 02/04/2018 | Desarrollador |
| 6 | Documento Final | 03/04/2018 | 09/04/2018 | Desarrollador |
| 7 | **Desarrollo del Juego** | **19/02/2018** | **23/03/2018** | |
| 8 | Estructura Básica | 19/02/2018 | 23/02/2018 | Desarrollador |
| 9 | Animaciones | 23/02/2018 | 26/02/2018 | Desarrollador y Artista |
| 10 | Entidades | 26/02/2018 | 05/03/2018 | Desarrollador |
| 11 | Controles | 06/03/2018 | 13/03/2018 | Desarrollador |
| 12 | Colisiones | 14/03/2018 | 21/03/2018 | Desarrollador |
| 13 | Textos | 27/02/2018 | 23/03/2018 | Desarrollador |
| 14 | **Desarrollo de la Inteligencia Artificial** | **01/03/2018** | **28/05/2018** | |
| 15 | Estudio y Aprendizaje | 01/03/2018 | 16/04/2018 | Desarrollador |
| 16 | Diseño e Implementación | 17/04/2018 | 07/05/2018 | Desarrollador |
| 17 | Pruebas y Verificación | 08/05/2018 | 28/05/2018 | Beta-testers |
| 18 | **Recolección y Evaluación de Datos** | **29/05/2018** | **12/06/2018** | |
| 19 | Recolección | 29/05/2018 | 06/06/2018 | Público |
| 20 | Evaluación | 07/06/2018 | 12/06/2018 | Desarrollador |
| 21 | **Hito Final** | **13/06/2018** | **29/06/2018** | |
| 22 | Finalizar Proyecto | 13/06/2018 | 22/06/2018 | Desarrollador |
| 23 | Redactar Memoria y Documentación | 13/06/2018 | 22/06/2018 | Desarrollador |
| 24 | Lectura y Defensa | 25/06/2018 | 29/06/2018 | Desarrollador |




## Valoración de Alternativas y Plan de Acción

Durante la realización del proyecto pueden surgir contratiempos o errores, si bien la metodología que queremos aplicar intenta valorar estos riesgos existe la posibilidad de que se deba modificar la organización con tal de solventar un fallo crítico.

Lo primero a realizar ante un imprevisto es localizar el fallo, para ello es necesario disponer de herramientas de rastreo en el código, como imprimir información relevante a través de la pantalla, de modo que podamos localizar el punto en que se genera el funcionamiento erróneo. Una vez se ha encontrado el fallo debemos analizar el código para poder saber que es lo que lo causa y así solventarlo. Si se trata de errores de código el coste de la reparación no suele ser importante, por lo que no causa desviaciones particulares en la planificación. Por otro lado, si el error es de lógica puede ser necesario tener que revisar partes implementadas anteriormente, por ello es necesario asegurarse que el fragmento en el que se está trabajando funciona de manera correcta antes de continuar con el ciclo de desarrollo. Cuando surgen estos problemas será necesario valorar el coste de la solución, muchas veces el desarrollar algo más eficiente puede ser más complejo a nivel de código, por lo que se deben comparar las ventajas y desventajas entre una solución simple u otra más eficiente. Es vital considerar la disponibilidad del desarrollador con estas cuestiones, ya que si hay tiempo libre que puede ser dedicado a trabajar en el proyecto es posible desarrollar la mejor solución a nivel técnico, pero que conlleva mayor coste de tiempo, sin afectar a la fecha de entrega. Por el contrario, existe la posibilidad que solo se disponga del tiempo medio estimado para dedicar al proyecto, con lo que será necesario apuntar por la solución más fácil de implementar. En general esta será la metodología a seguir en caso de desviaciones.

Puede darse el caso de que los imprevistos consuman demasiado tiempo, con lo que será necesario enfocar los recursos en las tareas más prioritarias y reducir el tiempo de otros aspectos menos importantes, incluso pudiendo llegar a tener que remover alguna de esas funcionalidades, como por ejemplo: los textos animados del juego. La idea es poder finalizar con un prototipo jugable que tenga el sistema de inteligencia artificial con la capacidad de aprendizaje.

Otras de las cuestiones que pueden surgir sería la falta de los recursos gráficos, es decir el material proporcionado por el artista. Si se da el caso el desarrollador puede recurrir a utilizar elementos del motor del juego (como las cajas contenedoras) a modo de indicadores visuales, ya que éstas están incluidas en el entorno aunque no se dibujan explícitamente. Esto serviría como una solución simple y eficiente en cuestión de tiempo, por lo que no causaría problemas para cumplir con las fechas de entrega.

## Recursos del Proyecto

A continuación se realiza un listado de los distintos recursos, tanto a nivel de hardware como de software, que se utilizarán en el desarrollo.

### Hardware

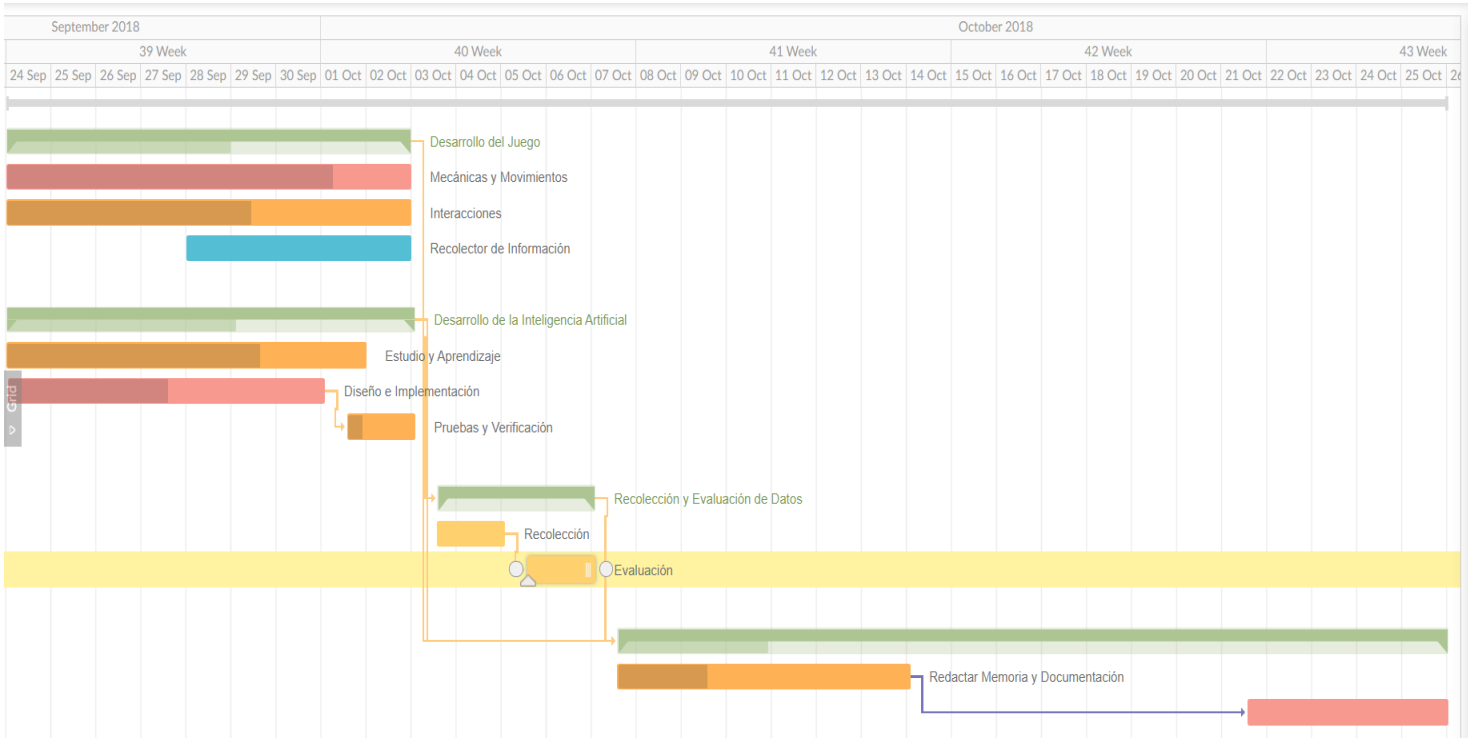
- Ordenador Portátil Asus para desarrollo y pruebas del proyecto

### Software

- Sistema Operativo Windows 10
- SFML 2.4.2 para la implementación del motor del juego
- Microsoft Visual Studio 2017 como entorno de programación
- Tortoise Git para sincronizar el proyecto con el repositorio online
- Notepad++ para anotaciones de ideas o planificación
- mlpack 2.2.5 para utilizar algoritmos ya implementados de aprendizaje automático (no está confirmado, pero se mantiene entre las opciones potenciales)
- Open Office 4.1.4 para la redacción de la memoria y los documentos

# Cambios en la Planificación

| Task name | Start date | End date | Duration week | Progress | Estimation (hours) | Assigned |
|---|---|---|---|---|---|---|
| *Total estimate* | *2018/09/24* | *2018/10/26* | *4.58* | | *922* | |
| **Desarrollo del Juego** | **2018/09/24** | 2018/10/03 | 1.29 | 55% | 276 | |
| Mecánicas y Movimientos | 2018/09/24 | 2018/10/03 | 1.29 | 80% | 108 | CA G |
| Interacciones | 2018/09/24 | 2018/10/03 | 1.29 | 60% | 108 | CA G |
| Recolector de Información | 2018/09/28 | 2018/10/03 | 0.71 | 0% | 60 | CA G |
| **Desarrollo de la Inteligencia Artificial** | **2018/09/24** | 2018/10/03 | 1.31 | 56% | 455 | |
| Estudio y Aprendizaje | 2018/09/24 | 2018/10/02 | 1.14 | 70% | 96 | CA G |
| Diseño e Implementación | 2018/09/24 | 2018/10/01 | 1.01 | 50% | 335.32 | CA G |
| Pruebas y Verificación | 2018/10/01 | 2018/10/03 | 0.29 | 20% | 24 | CA G |
| **Recolección y Evaluación de Datos** | 2018/10/03 | 2018/10/07 | 0.57 | 0% | 48 | |
| Recolección | 2018/10/03 | 2018/10/05 | 0.29 | 0% | 24 | CA G |
| Evaluación | 2018/10/05 | 2018/10/07 | 0.29 | 0% | 24 | CA G |
| **Hito Final** | 2018/10/07 | 2018/10/26 | 2.7 | 18% | 143 | |
| Redactar Memoria y Documentación | 2018/10/07 | 2018/10/14 | 1 | 30% | 84 | CA G |
| Lectura y Defensa | 2018/10/21 | 2018/10/26 | 0.7 | 0% | 59 | CA G |




En esta nueva planificación se valora todo lo que aún necesita ser trabajado. Se puede ver que el desarrollo del juego continuará en conjunto con el desarrollo de la IA, ya que algunos movimientos e interacciones podrán implementarse y refinarse en conjunto con las pruebas del comportamiento de los modelos de IA; también puede verse la adición de funcionalidades para recolectar información relevante de lo que ocurre en el juego y que será

necesaria para evaluar y comparar los modelos.

Respecto al desarrollo de la Inteligencia Artificial se tiene un progreso medio, se tiene el modelo básico y las bases del modelo con capacidad de aprendizaje. Hace falta diseñar e implementar el segundo modelo (IA con scripting y acciones más complejas) y lo restante del tercero (IA con aprendizaje).

Finalmente se realizarán pruebas con varios testers para recolectar información, para luego evaluarla y comparar los resultados. Se dedicará una semana para realizar la redacción de toda la memoria y la documentación del proyecto, para poder entregarse una semana antes de la fecha de la defensa oral (semana del 22 al 26 de Octubre).

## 8.2 Gestión Económica y Sostenibilidad

## Auto-evaluación sobre la Sostenibilidad

Tras la realización del cuestionario de estudiantes de Ingeniería Informática he sido capaz de valorar los conocimientos que he adquirido, en el transcurso de los estudios del Grado en Ingeniería Informática, acerca del tema de la sostenibilidad en los proyectos de Tecnologías de la Información y Comunicación. A través de las preguntas se me ha hecho posible plantearme cuanto sabía acerca de los distintos aspectos de la sostenibilidad, lo que también me permitió hacer retrospectiva y pensar en cuanto conocimiento sobre este tema adquirí en el transcurso de mis estudios del Grado.

Al comenzar los estudios no conocía el concepto de sostenibilidad, lo más aproximado era la consideración, por separado, del impacto ambiental y social que podía tener un proyecto. Aún así esas consideraciones no eran muy profundas y, en general, omitían puntos relevantes, como el impacto a largo plazo en sociedades o en el mundo profesional. En el Grado de Ingeniería Informática se nos enseñó el concepto y la importancia del desarrollo sostenible, considerando las tres dimensiones: ambiental, social y económica. Dentro de las asignaturas obligatorias se hizo énfasis en los aspectos antes mencionados, mientras que en algunas de carácter optativo pude aprender más detalles de importancia acerca de la sostenibilidad, tales como el uso de herramientas de trabajo colaborativo, metodologías y tecnologías, así como el conocimiento y aplicación de los principios de-ontológicos (como los que plantea el Col·legi Oficial d'Enginyeria Informàtica de Catalunya). Aún así considero que siguen habiendo algunos aspectos, dentro del campo de la sostenibilidad, que no han podido trabajarse de la manera adecuada, mayoritariamente por la especialidad y las asignaturas matriculadas..

De manera final, estoy seguro de que el Grado en Ingeniería Informática me ha concienciado y proporcionado un conocimiento muy valioso sobre la sostenibilidad en los proyectos de la actualidad. Aún así, con la encuesta, he podido comprobar que siguen habiendo algunos aspectos, de cierta relevancia para la planificación y desarrollo de proyectos, en los que no he sido formado con tanta profundidad.

# Análisis de Sostenibilidad del Proyecto

## Dimensión Económica

### Presupuesto

#### Identificación y Estimación de Costes

Es necesario realizar un presupuesto para poder estimar el coste total de este proyecto, para ello se hará un análisis de los distintos recursos necesarios para el desarrollo. Todos los elementos que se podrán ver son utilizados en las distintas actividades que fueron definidas en el diagrama de Gantt en el que se define la planificación temporal del trabajo.

El primer tipo de recurso a considerar serán las personas que participarán activamente en el desarrollo. En primer lugar se encuentra mi persona, la cual desempeñará el papel de desarrollador, encargado y beta-tester. En segundo lugar se encuentra Marcelo Nahuel Goncevatt, quien es el encargado del diseño y arte, así como también será beta-tester. Por último se hará mención del público al que se le pedirá participación sin remuneración, completamente voluntaria, para poder realizar pruebas finales y recolectar datos.

| Rol | Horas | Precio por Hora | Precio Total |
|---|---|---|---|
| Encargado | 130 | 30 €/h | 3.900€ |
| Desarrollador | 310 | 20 €/h | 6.200€ |
| Diseñador y Artista | 310 | 20 €/h | 6.200€ |
| Beta-tester | 80 | 25 €/h | 2.000€ |

Ya considerado el coste de los recursos humanos pasaremos a estimar el coste del hardware y el software. Es necesario aclarar que los programas y dispositivos que utilizará el diseñador y artista no serán incluidos en el listado, ya que esos costes serán cubiertos por la persona en cuestión. Primero evaluaremos el coste del único elemento de hardware que se mencionó en la planificación: el ordenador portátil Asus. Este dispositivo será utilizado para llevar a cabo toda la fase de desarrollo del proyecto. Podemos incluir una cantidad equivalente al 20% del precio del hardware para el caso de que surja un imprevisto y sea necesario cubrir costes de reparación, lo que equivaldría a unos 300€. El porcentaje es resultado de una estimación, considerando elementos como la necesidad de transportar el portátil fuera del entorno de trabajo para usos no relacionados al proyecto.

| Producto | Precio | Unidades | Vida Útil | Amortización |
|---|---|---|---|---|
| Asus ROG Strix GL502VM-FY213T | 1499€ | 1 | 5 años | 113,97€ |

Lo próximo es estimar el coste que pueden conllevar los programas utilizados, todos los elementos pertenecerán a la correspondiente lista de software detallada en la sección de Planificación Temporal. Lo primero que podemos encontrar es el sistema operativo sobre el cual trabajaremos: Windows 10 Home. Este software viene instalado en el portátil que utilizaremos, además proporciona compatibilidad con todos los demás programas y elementos software necesarios para el desarrollo del proyecto. La biblioteca SFML 2.4.2 es de acceso libre y gratuito, será utilizado para implementar el motor del juego. El entorno de desarrollo será Microsoft Visual Studio Community 2017, el cual no tiene coste ya que se ofrece de forma gratuita para estudiantes y desarrolladores, individuales y de código abierto. Tortoise Git será nuestro programa con interfaz gráfica para gestionar el repositorio el cual utilizaremos para guardar el proyecto y realizar un control y seguimiento de versiones, también es de acceso gratuito. Notepad++ es un editor de texto con varias funcionalidades que lo hacen práctico para realizar revisión de código o anotaciones de interés, no tiene coste de acceso o uso. Mlpack es una biblioteca con algoritmos de aprendizaje automático, es gratuita y es necesario considerarla ya que puede proporcionar ejemplos o funciones de utilidad. Finalmente tenemos Open Office 4.1.4, un editor de documentos de acceso gratuito el cual usaremos para realizar la redacción de la memoria y otra documentación necesaria para el proyecto. Es práctico realizar la observación de que todo el software que utilizaremos en este proyecto no tiene costes.

| **Producto** | **Precio** | **Unidades** | **Vida Útil** | **Amortización** |
|---|---|---|---|---|
| Windows 10 Home | 0€ | 1 | 5 años | 0€ |
| SFML 2.4.2 | 0€ | 1 | - | 0€ |
| Microsoft Visual Studio Community 2017 | 0€ | 1 | - | 0€ |
| Tortoise Git | 0€ | 1 | - | 0€ |
| Notepad++ | 0€ | 1 | - | 0€ |
| Mlpack 2.2.5 | 0€ | 1 | - | 0€ |
| Open Office 4.1.4 | 0€ | 1 | - | 0€ |

No hay otros gastos que sea necesario considerar para el desarrollo, por lo que podemos resumir el presupuesto total de la siguiente manera;

| **Concepto** | **Total** |
|---|---|
| Recursos Humanos | 18.300€ |
| Hardware | 113,97€ |
| Software | 0€ |

| **Presupuesto Total** | 18.413,97€ |
|---|---|

Podemos ver que tenemos un presupuesto viable, además podemos considerar un porcentaje aproximado del 15% como contingencia, de modo que tengamos un margen de 2.762,10€ para dedicar a otros fallos o descuidos que surjan durante el desarrollo.

### Control de Gestión

El mayor problema que puede afectar al presupuesto que hemos estimado sería la necesidad de dedicar más horas en algunas de las tareas planificadas para el proyecto, nos estamos refiriendo a un posible desvío en eficiencia. La única manera que tenemos de poder controlar este tipo de cuestión sería con la reorganización de algunas de las tareas, de modo que podamos terminar el desarrollo en el tiempo estimado. Aún así se dispone de un margen en el presupuesto para cubrir la necesidad de dedicar más horas, siempre y cuando se logre finalizar en la fecha final prevista. Otra posibilidad sería la de negociar con los actores pertinentes un ajuste en el coste, de modo que podamos adecuarnos al presupuesto final estimado.

## Reflexión

El coste del presupuesto estimado es asequible para lo que conlleva este tipo de proyectos hoy en día, la gran mayoría de la inversión estará dedicada a recompensar el esfuerzo y dedicación de quienes deben llevar a cabo el desarrollo, tanto a nivel técnico como artístico. El coste del hardware es bastante reducido y el del software es nulo, gracias al uso de recursos de acceso gratuito o que ya vienen incluidos con algún otro servicio.

Para poder realizar una comparación con el coste de trabajos de índole similar es necesario considerar el objetivo de los mismos, ya que el coste de un videojuego pensado para la comercialización masiva no será igual al de uno más simple, con menos expectativas de ventas, pero que intente llegar a un determinado público. El segundo tipo sería más cercano a lo que queremos desarrollar, de la misma forma tenemos un objetivo enfocado en crear un producto que agrade a los jugadores, los desafíe, y nos permita recolectar datos para evaluar el rendimiento de la aplicación del aprendizaje automático aplicado a la inteligencia artificial de un videojuego. Comparado con el coste de desarrollar un juego en el entorno independiente o “indie” el presupuesto que tenemos es bastante cercano, el coste vendrá dado principalmente por lo que cobre el personal y el precio del equipamiento para trabajar. Es posible decir que este proyecto costará menos dinero que la media de este ámbito, ya que el software que utilizamos es gratuito y los elementos a nivel artístico o de diseño serán creados por nosotros mismos, lo cual elimina la necesidad de pagar por ciertos programas, licencias o recursos artísticos.

## Dimensión Social

En lo que respecta al nivel personal creo que este proyecto me permitirá introducirme y trabajar en el campo de los videojuegos, el cual hoy en día ha ganado mucha importancia e impacto, así como es el tipo de trabajos en el cual busco especializarme. También me permitirá experimentar y aprender sobre técnicas orientadas a la inteligencia artificial, más

específicamente al ámbito del aprendizaje automático, otro tema que también me atrae bastante y puede ser combinado con el anterior. Sirve como un medio para adquirir experiencia en este tipo de trabajos.

Hoy en día la aplicación del aprendizaje automático en el sistema de inteligencia artificial de los videojuegos no es una práctica muy extendida, los experimentos y proyectos que combinan estos temas están más orientados en enseñar a una IA a jugar uno o más juegos de distintas complejidades. Aún así, hoy en día, existen propuestas y proyectos que buscan crear comportamientos que se adapten al jugador, ya sea a su nivel o su estilo de juego, permitiendo una mejor experiencia, sea a través del desafío o de la variación en el comportamiento, y un mayor nivel de inmersión en el mundo y la historia que se presentan. Creo que este proyecto se enfoca en trabajar esos aspectos, introducir en un juego una historia interesante y poco usual, un sistema de combate rápido y varios tipos de enemigos dinámicos que puedan adaptarse al estilo de la persona que participa. De esta forma el jugador podrá disfrutar, aprender, reflexionar, y mucho más, el videojuego le aportará medios y recursos que, por ejemplo, pueden hacerlo crecer como persona, o mejorar su estado de ánimo.

Creo que si existe una necesidad para este tipo de proyecto, principalmente por la cuestión de aplicar técnicas de aprendizaje automático a la inteligencia artificial del juego. Esto se debe a que, como se comentó antes, existen algunas propuestas y proyectos de este tipo, pero aún hay mucho espacio para realizar pruebas o probar distintos algoritmos, de modo que se pueda evaluar la eficacia y viabilidad de aplicar dichas técnicas para mejorar la experiencia del jugador y la calidad del propio juego desarrollado.

## Dimensión Ambiental

Este proyecto se enfoca en el desarrollo de software, no hay ningún aspecto particular que pueda llegar a valorarse acerca del impacto medioambiental.

Matriz de Sostenibilidad

| | **PPP** |
|---|---|
| Ambiental | Si, el impacto no es particularmente notable, lo que más puede considerarse es el consumo de recursos que tenga la aplicación en un ordenador (ya que mientras más recursos consuma, más trabajo necesitan realizar elementos como el procesador). Lo ideal sería crear una aplicación lo más eficiente posible para reducir ese consumo, para ello se deben implementar de manera adecuada los algoritmos y utilizar las estructuras de datos correspondientes. |
| Económico | He realizado una estimación de los recursos que serán necesarios, tanto a nivel humano como de material (hardware y software). |
| Social | Considero que este proyecto me aportará una experiencia y conocimientos importantes. En primer lugar servirá como una introducción al desarrollo e industria del videojuego. En segundo lugar me permitirá aprender y profundizar conocimientos relacionados a técnicas de aprendizaje automático del campo de la inteligencia artificial y su aplicación en el campo de los videojuegos. |

# Bibliografía


[1] Wolf, Mark J. P. (2007). *"The Video Game Explosion: A History from PONG to Playstation and Beyond"*. Greenwood: Mark J. P. Wolf ed.

[2] (2004) *"Retro Gamer"*, Bournemouth: Imagine Publishing, 2004

[3] Franklin, Paul (2009) *"At 25, Tetris still eyeing growth"* [En línea] [Consultado: 4 de Marzo 2018] Disponible en Internet: < https://www.reuters.com/article/us-videogames-tetris/at-25-tetris-still-eyeing-growth-idUSTRE5510V020090602 >

[4] Hamari, Juho; Sjöblom, Max (2016) *"What is eSports and why do people watch it?"*, Internet Research, Vol. 27 Issue: 2, pp. 211-232, https://doi.org/10.1108/IntR-04-2016-0085 [En línea] Emerald Publishing Limited [Consultado: 4 de Marzo 2018] Disponible en Internet: < http://www.emeraldinsight.com/doi/abs/10.1108/IntR-04-2016-0085 >

[5] Graft, Kris (2015) *"When artificial intelligence in video games becomes... artificially intelligent"* [En línea] [Consultado: 4 de Marzo de 2018] Disponible en Internet: < https://www.gamasutra.com/view/news/253974/When_artificial_intelligence_in_video_games_becomesartificially_intelligent.php >

[6] N.Yannakakis, Georgios N.; Togelius, Julian (2018) *"Artificial Intelligence and Games"*, Springer, 2018 [En línea] [Consultado: 4 de Marzo 2018] Disponible en Internet: < http://gameaibook.org/book.pdf >

[7] Bakkes, Sander; Spronck, Pieter; van den Herik, Jaap (2008) *"Rapid Adaptation of Video Game AI"*, 2008 IEEE Symposium On Computational Intelligence and Games, https://doi.org/10.1109/CIG.2008.5035624 [En línea] [Consultado: 8 de Abril 2018] Disponible en Internet: < http://www.csse.uwa.edu.au/cig08/Proceedings/papers/8059.pdf >

[8] Ricciardi, Antonio; Thill, Patrick (2008) *"Adaptive AI for Fighting Games"* [En línea] [Consultado: 8 de Abril 2018] Disponible en Internet: < http://cs229.stanford.edu/proj2008/RicciardiThill-AdaptiveAIForFightingGames.pdf >

[9] Ram, Ashwin; Ontañón, Santiago; Mehta, Manish (2007) *"Artificial Intelligence for Adaptive Computer Games"*, 20th International FLAIRS Conference on Artificial Intelligence (FLAIRS-2007), AAAI Press [En línea] [Consultado: 8 de Abril 2018] Disponible en Internet: < https://www.cc.gatech.edu/faculty/ashwin/papers/er-07-04.pdf >

[10] Spronck, Pieter; Ponsen, Marc; Sprinkhuizen-Kuyper, Ida; Postma, Eric (2005) *"Adaptive Game AI with Dynamic Scripting"*, Machine Learning; Vol. 63 Issue 3, pp. 217 – 248, https://doi.org/10.1007/s10994-006-6205-6 [En línea] [Consultado: 8 de Abril 2018] Disponible en Internet: < http://www.spronck.net/pubs/DynamicScripting.pdf >

[11] Elkin, Aaron (2013) *"Video Game Satisfaction with Adaptive Game AI"* [En línea] [Consultado: 8 de Marzo 2018] Disponible en Internet: < http://orzo.union.edu/Archives/SeniorProjects/2013/CS.2013/files/elkina/elkina_paper.pdf >

[12] Gomila, Laurent (2018) *"Sitio web de SFML"* [En línea] [Consultado: 4 de Marzo 2018]

Disponible en Internet: < https://www.sfml-dev.org/index.php >

[13] Curtin, Ryan (2018) *"Sitio web de mlpack"* [En línea] [Consultado: 4 de Marzo 2018] Disponible en Internet: < https://www.mlpack.org/ >

[14] Atlassian (2018) *"Bitbucket"* [En línea] [Consultado: 8 de Abril 2018] Disponible en Internet: < https://bitbucket.org/product >

[15] Stunlock Studios (2016) *"Sitio web de Arena Battlerite"* [En línea] [Consultado: 12/10/2018] Disponible en Internet: < https://arena.battlerite.com/ >

[16] Square Enix (2017) *"Sitio web de NieR: Automata"* [En línea] [Consultado: 12/10/2018] Disponible en Internet: < https://www.niergame.com/ >

[17] Lab Zero Games (2012) *"Sitio web de Skullgirls"* [En línea] [Consultado: 12/10/2018] Disponible en Internet: < http://skullgirls.com/ >